CPL6 API Reference Manual

— DRAFT —

Tony Craig[1]
Brian G. Kauffman[1]
Robert Jacob[2]
Tom Bettge[1]
Jay Larson[2]

July 5, 2004

[1] National Center for Atmospheric Research, PO Box 3000, Boulder CO 80307
[2] Argonne National Laboratory, Argonne, IL

This document containts the Application Programmer's Interface (API) for the subroutines and functions contained in cpl6. It includes the general modules in `../models/csm_share/cpl` and also includes the modules developed to support the `main` in `../models/cpl/cpl6`.

Only the public subroutines and datatypes are included here.

This document was generated automatically from the Protex documentation in the source code.

Contents

Part I
General Modules and Datatypes

1 Domain

1.1 Module cpl_domain_mod – Grid and decomposition information (Source File: cpl_domain_mod.F90)

The *domain* data type is a fundamental coupler data type. A *domain* contains both physical information about a *grid* such as latitude and longitude values for the points as well as information about the *decomposition*. The decomposition is described by an MCT *GlobalSegMap* (gsMap).
NOTE: Currently there is no initialization routine in the module. *domains* are initialized during the *contract* initialization in `cpl_contract_init`.

REVISION HISTORY:

```
2001-aug-15 - B. Kauffman - created module
```

INTERFACE:

```
module cpl_domain_mod
```

USES:

```
use shr_sys_mod    ! shared system call wrappers
use cpl_kind_mod   ! kinds
use cpl_mct_mod    ! MCT API
use cpl_comm_mod   ! communicator groups, pids, etc.
use cpl_control_mod, only: dbug=>cpl_control_infoDBug

implicit none

private ! except
```

PUBLIC TYPES:

```
public :: cpl_domain

type cpl_domain
   !--- decomposition-independant data ---
   character(80) :: name   ! = "null" ! name of domain       (eg. "ocean")
   character(80) :: suffix ! = "null" ! netCDF domain suffix (eg. "o")
   integer(IN)   :: n      ! n = ni*nj ~ total number of grid pts (global)
   integer(IN)   :: ni     ! number of 2d array i indicies        (global)
   integer(IN)   :: nj     ! number of 2d array j indicies        (global)

   !--- decomposition-dependant data ---
   type(cpl_mct_aVect) :: lGrid ! grid data
   type(cpl_mct_gsMap) :: gsMap ! global seg map (defines decomp)
end type cpl_domain
```

PUBLIC MEMBER FUNCTIONS:

```
public cpl_domain_info     ! print some info about a domain
public cpl_domain_clean    ! clean/dealloc a domain
public cpl_domain_compare  ! compare two domains for consistency
```

PUBLIC DATA MEMBERS:

```
! no public data members
```

1.1.1 cpl_domain_info – Write info about domain.

Write basic information about the input domain `cpl_domain_x` to stdout. This information is useful for debugging.

REVISION HISTORY:

```
2001-Dec-20 - B. Kauffman -- first prototype
```

INTERFACE:

```
subroutine cpl_domain_info(cpl_domain_x)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_domain)   ,target,intent(in) :: cpl_domain_x  ! domain
```

1.1.2 cpl_domain_clean – Clean a domain type

Thie routine deallocates the allocated memory associated with the input/ouput `dom` argument.

REVISION HISTORY:

```
2002-Jan-20 - T. Craig -- first prototype
```

INTERFACE:

```
subroutine cpl_domain_clean(dom)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_domain)   ,intent(inout) :: dom  ! domain
```

1.1.3 cpl_domain_compare - Compare two domains

Compares two domains and summmarizes the differences. It compares the size, and also checks that the mask, and both model and mapping area are identical to within a `eps` factor defined below.
The various `enforce_*` optional arguments will, if present and true, force this routine to abort if the desired test fails.

REVISION HISTORY:

```
     2004-May-21 - B. Kauffman, initial version
```

INTERFACE:

```
 subroutine cpl_domain_compare(dom1,dom2,enforce_mask,enforce_grid, &
           &                           enforce_area,enforce_aream, enforce_all)
```

USES:

```
    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_domain),intent(in) :: dom1          ! domain #1
    type(cpl_domain),intent(in) :: dom2          ! domain #2
    logical,optional,intent(in) :: enforce_mask  ! abort if masks differ wrt zero/nonzero
    logical,optional,intent(in) :: enforce_grid  ! abort if grids differ by eps_grid
    logical,optional,intent(in) :: enforce_area  ! abort if area  differ by eps_area
    logical,optional,intent(in) :: enforce_aream ! abort if aream differ by eps_area
    logical,optional,intent(in) :: enforce_all   ! abort for all of the above
```

2 Bundle

2.1 Module cpl_bundle_mod – fundamental data type definition (Source File: cpl_bundle_mod.F90)

The *bundle* data type is a fundamental coupler data type. A *bundle* consists of one or more fields, all of which share the same *domain*. The field data is stored together in an MCT *AttributeVector* (`mct_aVect`) which provides a flexible and indexible storage type. Following MCT, the individual fields (T, u, SST, etc.) are referred to as "attributes". Each attribute contains the data for a field in a one-dimensional vector. The size of the vector is equal to the local size of the *domain* on a processor.
This module defines the *bundle* datatype and provides several methods for manipulating bundles.

REVISION HISTORY:

```
2002-Sep-10 - T. Craig - add cpl_bundle_split, cpl_bundle_gather
2001-Mar-20 - T. Craig, B. Kauffman, R. Jacob - first prototype
```

INTERFACE:

```
module cpl_bundle_mod
```

USES:

```
use cpl_mct_mod
use cpl_comm_mod
use cpl_domain_mod
use cpl_kind_mod
use cpl_control_mod, only: dbug=>cpl_control_infoDBug
use cpl_control_mod, only: bfbflag=>cpl_control_bfbflag
use shr_sys_mod
use shr_mpi_mod

implicit none

private ! except
```

PUBLIC TYPES:

```
public :: cpl_bundle

type cpl_bundle
   character(80)            :: name  ! id string for bundle
   type(cpl_mct_aVect)      :: data  ! attribute vector containing data
   type(cpl_domain),pointer :: dom   ! domain associated with data
   integer(IN)              :: cnt   ! counter for accumulating bundles
end type cpl_bundle
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_bundle_init
public :: cpl_bundle_initv
public :: cpl_bundle_clean
public :: cpl_bundle_info
public :: cpl_bundle_fill
public :: cpl_bundle_dump
public :: cpl_bundle_copy
public :: cpl_bundle_fcopy
public :: cpl_bundle_split
public :: cpl_bundle_gather
```

```
public :: cpl_bundle_hasAttr
public :: cpl_bundle_zero
public :: cpl_bundle_accum
public :: cpl_bundle_avg
public :: cpl_bundle_add
public :: cpl_bundle_mult
public :: cpl_bundle_divide
public :: cpl_bundle_gsum
```

PUBLIC DATA MEMBERS:

```
! no public data
```

2.1.1 cpl_bundle_init - Initialize the bundle data type

Initialize the bundle data type `bun`. `bun` will be given the name `name` and associated with the *domain* `dom`. Memory will be allocated to hold the attributes listed in the input `rList` in an MCT *AttributeVector*.

REVISION HISTORY:

```
2001-Mar-20  - T. Craig, B. Kauffman, R. Jacob - first prototype
```

INTERFACE:

```
subroutine cpl_bundle_init(bun,name,rList,dom)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(out)       :: bun   ! bundle to initialize
character(*)    ,intent(in)        :: name  ! name used in netCDF files
character(*)    ,intent(in)        :: rList ! aVect real data list string
type(cpl_domain),intent(in),target :: dom   ! domain assigned to bundle
```

2.1.2 cpl_bundle_initv - Initialize the bundle data type using another bundle

This routine will initialize `bun` to have the same attributes as the input `bun2` but will be given the new name `name` and will be associated with the *domain* `dom`. Note that this routine only allocates memory and does not copy the contents of `bun2` to `bun`.

REVISION HISTORY:

```
2002-jan-15 - T. Craig - first prototype
```

INTERFACE:

```
subroutine cpl_bundle_initv(bun,name,bun2,dom)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(out)       :: bun  ! bundle to initialize
character(*)    ,intent(in)        :: name ! name used in netCDF files
type(cpl_bundle),intent(in)        :: bun2 ! bundle to "copy" from
type(cpl_domain),intent(in),target :: dom  ! domain assigned to bundle
```

2.1.3 cpl_bundle_clean - Clean the bundle data type

Deallocate all allocated memory associated with the input/output argument `bun`.

REVISION HISTORY:

```
2002-Jan-20 - T. Craig - first prototype
```

INTERFACE:

```
subroutine cpl_bundle_clean(bun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)    ,intent(inout)     :: bun ! bundle to initialize
```

2.1.4 cpl_bundle_info - Print out bundle info.

Print out the following information about the bundle `bun`:

- the name
- the name of the associated *domain*
- the accumulation count
- the names of the attributes

REVISION HISTORY:

```
2002-May-09 - B. Kauffman - make's use of cpl_mct_aVect_info routine
2001-Jun-14 - T. Craig
```

INTERFACE:

```
subroutine cpl_bundle_info(bun)
```

USES:

```
use cpl_fields_mod
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)    ,intent(in)     :: bun ! bundle to initialize
```

2.1.5 cpl_bundle_fill - Fill a bundle with test data.

Fill each field in the input bundle bun with a slightly different sine wave. Used for debugging.

REVISION HISTORY:

2001-Jun-14 - T. Craig

INTERFACE:

```
subroutine cpl_bundle_fill(bun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout) :: bun   ! bundle to fill
```

2.1.6 cpl_bundle_dump - write bundle contents to a file.

Write out the contents of bun on each processor to a seperate file. Filename will be fort.(iun + coupler processor id).

REVISION HISTORY:

2002-Jan-14 - T. Craig

INTERFACE:

```
subroutine cpl_bundle_dump(iun,bun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(in) :: bun   ! bundle to write
integer(IN),intent(in)      :: iun   ! base unit number
```

2.1.7 cpl_bundle_copy - copy data from one bundle to another

This routine copies from input argument inbun into the output argument outbun the data of all the attributes shared between the two. If only a subset of shared attributes should be copied, use the optional arguments bunrList and buniList to specify which attributes should be copied. If any attributes in outbun have different names for the same quantity, provide a corresponding optional translation list, bunTrList and bunTiList, in addition to bunrList and buniList. The translation list is identical to the input List with translated names in place of the names in inbun.
Optional argument fcopy directs this routine to use cpl_bundle_copy instead of cpl_mct_aVect_copy. This is obsolete since cpl_mct_aVect_copy now uses the same algorithm.

REVISION HISTORY:

2002-Jul-02 - R. Jacob -- initial version

INTERFACE:

```
subroutine cpl_bundle_copy(inbun,bunrList,bunTrList,buniList,bunTiList,outbun,fcopy)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_bundle),intent(in)          :: inbun    ! bundle to read
    character(*)    ,intent(in),optional :: buniList
    character(*)    ,intent(in),optional :: bunrList
    character(*)    ,intent(in),optional :: bunTiList
    character(*)    ,intent(in),optional :: bunTrList
    type(cpl_bundle),intent(out)         :: outbun    ! bundle to write to
    logical         ,intent(in),optional :: fcopy     ! use fcopy
```

2.1.8 cpl_bundle_fcopy - Fast copy version of cpl_bundle_copy. Obsolete.

This routine copies from input argument `inbun` into the output argument `outbun` the data of all the attributes shared real between the two.
NOTE: This routine will be deleted in future versions since it is no longer necessary.

REVISION HISTORY:

```
       2003-Aug-26 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_fcopy(inbun,outbun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_bundle),intent(in)          :: inbun    ! bundle to read
    type(cpl_bundle),intent(out)         :: outbun    ! bundle to write to
```

2.1.9 cpl_bundle_split - Split bundle into multiple bundles.

Copy data from input bundle `bun_X` into multiple output bundles. Can have up to 12 output bundles: `bun1`, `bun2`, `bun3` etc. On return, each output bundle with an attribute name that matches an attribute in `bun_X` will have a copy of the data for that attribute.
The optional argument `fcopy` will use the fast version of cpl_bundle_copy.
This routine is used to "split" data in a bundle received in the Coupler from a model into bundles for each data pathway through the Coupler.
NOTE: All arguments must already be initialized.

REVISION HISTORY:

```
       2002-Apr-12 - B. Kauffman - first version
       2002-Jul-02 - R. Jacob - use bundle copy
       2002-Jul-15 - T. Craig - generalized
```

INTERFACE:

```
subroutine cpl_bundle_split(bun_X,bun1,bun2,bun3,bun4,bun5,bun6,bun7,bun8,bun9,bun10,bun11,bun12,fco
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),         intent(in ) :: bun_X ! input bundle
type(cpl_bundle),optional,intent(out) :: bun1  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun2  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun3  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun4  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun5  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun6  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun7  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun8  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun9  ! split bundle
type(cpl_bundle),optional,intent(out) :: bun10 ! split bundle
type(cpl_bundle),optional,intent(out) :: bun11 ! split bundle
type(cpl_bundle),optional,intent(out) :: bun12 ! split bundle
logical         ,optional,intent(in)  :: fcopy ! use fcopy
```

2.1.10 cpl_bundle_gather - Copy data into one bundle from many

Copy into the output argument `bun_X` all the data from the input bundles `bun1, bun2, bun3` etc. which have the same attribute names.
The optional argument `fcopy` will use the fast version of cpl_bundle_copy.
If there is a common attribute name between 2 or more of the input bundles which also exists in the output bundle, the values in the output bundle will that of the last listed input bundle.
This routine is used to "gather" data from multiple sources into one bundle for sending from the Coupler to a model.
NOTE: All arguments must already be initialized.

REVISION HISTORY:

```
2002-Jun-22 - B. Kauffman - first version
2002-Jul-15 - T. Craig - generalized
2003-Sep-01 - T. Craig - add fcopy optional argument
```

INTERFACE:

```
subroutine cpl_bundle_gather(bun_X, bun1,bun2,bun3,bun4,bun5,bun6,bun7,bun8,bun9,bun10,bun11,bun12,f
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),         intent(out) :: bun_X ! output bundle
type(cpl_bundle),optional,intent(in ) :: bun1  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun2  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun3  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun4  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun5  ! gather bundle
```

```
type(cpl_bundle),optional,intent(in ) :: bun6  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun7  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun8  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun9  ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun10 ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun11 ! gather bundle
type(cpl_bundle),optional,intent(in ) :: bun12 ! gather bundle
logical         ,optional,intent(in)  :: fcopy ! use fcopy
```

2.1.11 cpl_bundle_hasAttr

Return true if input bundle `bun` has any real or integer attributes.

REVISION HISTORY:

```
2002-Sep-11  - R. Jacob - first version
```

INTERFACE:

```
logical function cpl_bundle_hasAttr(bun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(in ) :: bun
```

2.1.12 cpl_bundle_zero - Zero values of fields in bundle

Set all fields (real and integer) in input bundle `bun` to 0. If the optional `fld` argument is present, only set that field to 0.
The bundle must already be initialized.

REVISION HISTORY:

```
2002-Sep-15 - T. Craig -- initial version
2002-Sep-17 - T. Craig -- add optional fld argument
```

INTERFACE:

```
subroutine cpl_bundle_zero(bun,fld)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)     ,intent(inout) :: bun   ! bundle to zero
character(*),optional,intent(in)    :: fld   ! field in bundle to zero
```

2.1.13 cpl_bundle_accum - Accumulate fields in a bundle.

Accumulate fields in input bundle `inbun` in output bundle `outbun`. This mimics cpl_bundle_copy but instead of overwriting the data in `outbun` with the data in `inbun`, the data in `inbun` is added to the corresponding fields in `outbun`.
It is recommended that a cpl_bundle_accum only be called once for each set of fields to be accumulated at each accumulation step because of the primitive nature of the bundle counter (buntime cpl_bundle_accum is called for the output bundle.

REVISION HISTORY:

```
2002-Sep-15 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_accum(inbun,bunrList,bunTrList,buniList,bunTiList,outbun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(in)          :: inbun   ! bundle to read
character(*)    ,intent(in),optional :: buniList
character(*)    ,intent(in),optional :: bunrList
character(*)    ,intent(in),optional :: bunTiList
character(*)    ,intent(in),optional :: bunTrList
type(cpl_bundle),intent(out)         :: outbun   ! bundle to write to
```

2.1.14 cpl_bundle_avg - averages a bundle

Average the data in input bundle `bun`. Divide all fields in the bundle `bun` by the value of the bundle counter, `bun%cnt`.

REVISION HISTORY:

```
2002-Sep-15 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_avg(bun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout) :: bun   ! bundle to read
```

2.1.15 cpl_bundle_add - add product of multiple bundles.

Form the product of individual fields from the input bundles `bun1, bun2, bun3` etc. and *add* it to the field `fld` in the input/output argument `bun`.
`fld1` is taken from `bun1`, `fld2` is taken from `bun2`, etc.
If only `fld1` and `bun1` are present, then `fld1` will be added to `bun`
If the optional argument `scalar` is present, it will also by part of the product.
If the optional argument `zero` is present, the input/output argument `bun` will be zeroed before adding the product.
This routine is used to *merge* data in the Coupler.

REVISION HISTORY:

```
2002-Sep-15 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_add(bun,fld,bun1,fld1,bun2,fld2,bun3,fld3,bun4,fld4,bun5,fld5,bun6,fld6,scalar
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)         ,intent(inout) :: bun    ! bundle output
character(*)             ,intent(in)    :: fld    ! bun  field name
type(cpl_bundle)         ,intent(in)    :: bun1   ! bundle input
character(*)             ,intent(in)    :: fld1   ! bun1 field name
type(cpl_bundle),optional,intent(in)    :: bun2   ! bundle input
character(*)    ,optional,intent(in)    :: fld2   ! bun2 field name
type(cpl_bundle),optional,intent(in)    :: bun3   ! bundle input
character(*)    ,optional,intent(in)    :: fld3   ! bun3 field name
type(cpl_bundle),optional,intent(in)    :: bun4   ! bundle input
character(*)    ,optional,intent(in)    :: fld4   ! bun4 field name
type(cpl_bundle),optional,intent(in)    :: bun5   ! bundle input
character(*)    ,optional,intent(in)    :: fld5   ! bun5 field name
type(cpl_bundle),optional,intent(in)    :: bun6   ! bundle input
character(*)    ,optional,intent(in)    :: fld6   ! bun6 field name
real(R8)        ,optional,intent(in)    :: scalar ! scalar multiplier
logical         ,optional,intent(in)    :: zero   ! zero lhs initially
```

2.1.16 cpl_bundle_mult - multiply a bundle by a field.

Replace each field in `bun` by the product of that field and the field `fld1` from input argument `bun1`.
If optional argument `bunlist` is present, only those attributes in `bun` will be replaced.
If optional argument `initbun` is present, then the data in `bun` is replaced by the product of the data in `initbun` and `fld1` from `bun1`. NOTE: this assume `initbun` has the exact same attributes in the same order as `bun`.

REVISION HISTORY:

```
2002-Oct-15 - T. Craig -- initial version
2003-Jan-02 - T. Craig -- added bundle sub-list option (bunlist)
```

INTERFACE:

```
subroutine cpl_bundle_mult(bun,bun1,fld1,bunlist,initbun)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)     ,intent(inout) :: bun    ! bundle output
type(cpl_bundle)     ,intent(in)    :: bun1   ! bundle input
character(*)         ,intent(in)    :: fld1   ! bun1 field name
character(*),optional,intent(in)    :: bunlist! sublist of field in bun
type(cpl_bundle),optional,intent(in):: initbun! optional initialization bun
```

2.1.17 cpl_bundle_divide - Divide a bundle by a field

Divide each attribute in `bun` by the data in `fld1` from the input bundle `bun1`

REVISION HISTORY:

```
2002-Oct-15 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_divide(bun,bun1,fld1)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout) :: bun    ! bundle output
type(cpl_bundle),intent(in)    :: bun1   ! bundle input
character(*)    ,intent(in)    :: fld1   ! bun1 field name
```

2.1.18 cpl_bundle_gsum - Calculate global sum and output

Calculate the global sum of all the attributes in input bundle `bun` and output the results to stdout.
The data in `bun` can be weighted by the optional arguments. If present, the data in `bun` will be multiplied by the data from `fld1` of `AV1` and data from `fld2` of `AV2` and the scalar `scalar`.
If optional argument `istr` is present, it will be included in the output string between the global sum and the name of the attribute.

REVISION HISTORY:

```
2003-Jan-2 - T. Craig -- initial version
```

INTERFACE:

```
subroutine cpl_bundle_gsum(bun,AV1,fld1,AV2,fld2,scalar,istr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle)            ,intent(in)    :: bun    ! bundle input
type(cpl_mct_aVect),optional,intent(in)    :: AV1    ! weight bundle input
character(*)       ,optional,intent(in)    :: fld1   ! AV1 field name
type(cpl_mct_aVect),optional,intent(in)    :: AV2    ! weight bundle input
character(*)       ,optional,intent(in)    :: fld2   ! AV2 field name
real(R8)           ,optional,intent(in)    :: scalar ! scalar for weights
character(*)       ,optional,intent(in)    :: istr   ! string for print
```

3 InfoBuffer

3.1 Module cpl_infobuf_mod – information buffer module (Source File: cpl_infobuf_mo

The cpl6 infobuf, or "information buffer", is used to exchange control flags and other miscellaneous non-gridded control information that is typically sent/received along with gridded field data.
Currently the infobuf is simply two arrays, one real and one integer.

REMARKS:

```
The infobuf is exchanged between a model and component with
each communication call.
```

REVISION HISTORY:

```
2002-Dec-5  - T. Craig - Moved cpl_coupling_ibuf methods here.
2003-Jan-10 - R. Jacob - change this module to work with an infobuf type.
2003-Jan-15 - T. Craig - Renamed this to infobuf from ibuf
```

INTERFACE:

```
module cpl_infobuf_mod
```

USES:

```
use cpl_kind_mod
use cpl_fields_mod
use shr_timer_mod
use shr_sys_mod
use shr_mpi_mod

implicit none

private ! except
```

PUBLIC TYPES:

```
integer(IN),parameter,public :: cpl_infobuf_ibufSize = cpl_fields_ibuf_total
integer(IN),parameter,public :: cpl_infobuf_rbufSize = cpl_fields_rbuf_total

public :: cpl_infobuf

type cpl_infobuf
   integer(IN) :: ibuf(cpl_infobuf_ibufSize) ! integer data
   real(R8)    :: rbuf(cpl_infobuf_rbufSize) ! real    data
end type cpl_infobuf
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_infobuf_init   ! initialize infobuf to default values
public :: cpl_infobuf_send   ! send an infobuf
public :: cpl_infobuf_recv   ! recv an infobuf
public :: cpl_infobuf_bcast  ! broadcast an infobuf
```

PUBLIC DATA MEMBERS:

```
integer(IN),parameter,public :: cpl_infobuf_iDefault = 0
integer(IN),parameter,public :: cpl_infobuf_rDefault = 0.0
!integer(IN),parameter,public :: cpl_infobuf_ibufSize = ! must define above
!integer(IN),parameter,public :: cpl_infobuf_rbufSize = ! must define above
```

3.1.1 cpl_infobuf_init – initialize to default values

Initialize the input `infobuf` to default values. The integers are initialized to cpl_infobuf_iDefault while the reals are intitialized to cpl_infobuf_rDefault

REVISION HISTORY:

```
2003-Jan-15 - B. Kauffman -- initial version.
```

INTERFACE:

```
subroutine cpl_infobuf_init(infobuf)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_infobuf), intent(out):: infobuf   ! info buffer
```

3.1.2 cpl_infobuf_send – Send an infobuf

Send contents of `infobuf` array to processor `pid` within the *MPI_ communicator* `comm` using `tag` to identify the message.

REVISION HISTORY:

```
2002-Aug-05 - T. Craig -- abstracted mpi_send call into subroutine
```

INTERFACE:

```
subroutine cpl_infobuf_send(infobuf,pid,tag,comm)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_infobuf), intent(inout):: infobuf                  ! info buffer
integer(IN), intent(in)     :: pid                  ! proc id
integer(IN), intent(in)     :: tag                  ! tag
integer(IN), intent(in)     :: comm                 ! mpi communicator
```

3.1.3 cpl_infobuf_recv – Receive an infobuf

Receive contents of `infobuf` array from processor `pid` within the *MPI_ communicator* `comm` using `tag` to identify the message.

REVISION HISTORY:

```
2002-Aug-05 - T. Craig -- abstracted mpi_recv call into subroutine
```

INTERFACE:

```
subroutine cpl_infobuf_recv(infobuf,pid,tag,comm)
```

USES:

```
    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_infobuf), intent(out):: infobuf                    ! info buffer
    integer(IN), intent(in)   :: pid                      ! proc id
    integer(IN), intent(in)   :: tag                      ! tag
    integer(IN), intent(in)   :: comm                     ! mpi communicator
```

3.1.4 cpl_infobuf_bcast – generic bcast of infobuf

Broadcast `infobuf` from processor `pid` to all processors on MPI_Communicator `comm`.

REVISION HISTORY:

```
    2002-Aug-05 - T. Craig -- abstracted mpi_bcast call into subroutine
```

INTERFACE:

```
subroutine cpl_infobuf_bcast(infobuf,pid,comm)
```

USES:

```
    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_infobuf), intent(inout) :: infobuf                 ! integer buffer
    integer(IN), intent(in)      :: pid                 ! proc id
    integer(IN)                  :: tag                 ! tag
    integer(IN), intent(in)      :: comm                ! mpi communicator
```

4 Contract

4.1 Module cpl_contract_mod – coupler/component contract type (Source File: cpl_contract_mod.F90)

The *contract* datatype encapsulates all the information needed to communicate data between the Coupler and a model. This includes both the information being exchanged, contained in a *bundle* and an *infobuffer* and the relevant *domain* for the *bundle*. The *contract* also contains the information needed to send the data between the model's and the coupler's processors in an MCT *Router* datatype.
The routines in this module initialize and process contracts. It's functionality relies heavily on the MCT, MPH, and MPI libraries. These routines should not be called directly, use the `cpl_interface_mod` routines instead.

REVISION HISTORY:

```
2002-Jul-16 - T. Craig - abstracted basic functionality from
              cpl_msg and cpl_interface to this layer.
2002 Aug 01 - T. Craig - prototype for contract datatype
2002 Dec 05 - T. Craig - combined cpl-coupling module and cpl_contract
```

INTERFACE:

```
module cpl_contract_mod
```

USES:

```
use shr_timer_mod       ! timers
use cpl_kind_mod        ! kinds
use cpl_mct_mod         ! mct interface
use cpl_comm_mod        ! mpi/mph communicator info
use cpl_fields_mod      ! fields module
use cpl_bundle_mod      ! defines bundle
use cpl_domain_mod      ! defines domain
use cpl_infobuf_mod     ! defines infobuf
use cpl_control_mod, only: dbug=>cpl_control_infoDBug

implicit none

private ! except
```

PUBLIC TYPES:

```
public :: cpl_contract

type cpl_contract
   type(cpl_infobuf)     :: infobuf       ! infobuf that goes with contract
   type(cpl_bundle)      :: bundle        ! bundle
   type(cpl_domain)      :: domain        ! domain info (grid with decomp)
   type(cpl_mct_Router)  :: rtr           ! MxN communication info
end type cpl_contract
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_contract_execute
public :: cpl_contract_send
public :: cpl_contract_recv
public :: cpl_contract_init
public :: cpl_contract_initSend
public :: cpl_contract_initRecv
```

PUBLIC DATA MEMBERS:

```
! none
```

4.1.1 cpl_contract_execute – send/recv data/msg to component.

Code for `cpl_contract_send` and `cpl_contract_recv` combined into one routine.

REMARKS:

REVISION HISTORY:

```
2002-09-10 - T.Craig - merged cpl_contract_send, cpl_contract_recv
```

INTERFACE:

```
subroutine cpl_contract_execute(srtype,contract,mypid,comm,otherpid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*)      ,  intent(in)   :: srtype   ! 'send' or 'recv'
type(cpl_contract),  intent(inout):: contract ! contract
integer(IN),         intent(in)   :: mypid    ! my mpi process ID
integer(IN),         intent(in)   :: comm     ! local communicator group
integer(IN),         intent(in)   :: otherpid ! mpi process ID to send to
```

4.1.2 cpl_contract_send – send data/msg to component.

Send the data contained in the *infobuffer* and *bundle* of the input `contract`. *infobuffer* is sent from the root of *MPI_Communicator* `comm` and sent to `otherpid` in *cpl_comm_wrld*.

REMARKS:

```
cpl_comm_wrld is defined in cpl_comm_mod
```

REVISION HISTORY:

```
2002-08-01 - T.Craig - abstracted from cpl_interface and cpl_msg
```

INTERFACE:

```
subroutine cpl_contract_send(contract,mypid,comm,otherpid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_contract),  intent(inout):: contract ! contract
integer(IN),         intent(in)   :: mypid    ! my mpi process ID
integer(IN),         intent(in)   :: comm     ! local communicator group
integer(IN),         intent(in)   :: otherpid ! mpi process ID to send to
```

4.1.3 cpl_contract_recv – receive data/msg from component.

Receive data into the *infobuffer* and *bundle* of the input/output argument `contract`. *infobuffer* is recived from `otherpid` in *cpl_comm_wrld* on the root of *MPI_Communicator* `comm` and then broadcast over `comm`

REMARKS:

```
cpl_comm_wrld is defined by cpl_comm_mod
```

REVISION HISTORY:

```
2002-08-01 - T.Craig - abstracted from cpl_interface and cpl_msg
```

INTERFACE:

```
subroutine cpl_contract_recv(contract,mypid,comm,otherpid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_contract),  intent(inout):: contract  ! contract
integer(IN),         intent(in)   :: mypid     ! my mpi process ID
integer(IN),         intent(in)   :: comm      ! local communicator group
integer(IN),         intent(in)   :: otherpid  ! mpi process ID to recv from
```

4.1.4 cpl_contract_init – Initialize a contract

Initialize the input `contract` and the *domain* in the `contract`. The contract will be between model `my_name` and model `ohter_name`. Both models must make matching calls to this routine.

This routine is currently somewhat uneven because it both initializes a contract *and* sends grid data (lat, lon, area) values from a model to the Coupler *and* initializes the *domain* in the *contract*. This is how the Coupler "learns" what grid each model is running on. (The Coupler does not know this information at runtime.) The Coupler always calls this routine with `srtype` set to "recv" while the models call it with `srtype` set to "send" and supply the optional argument `buf` which contains the grid data. The sequence of events is:

1. Model sends *infobuffer* to Coupler which contains total sizes of grid.
2. both Model and Coupler initialize total grid size portion of *contract*'s *domain*.
3. Coupler describes decomposition of Model's grid on the Coupler processors according to input argument `decomp`. Default decomposition is simple one-dimensional over all Coupler processors.
4. Both Model and Coupler initialize MCT *GlobalSegMap* part of *domain*
5. Model and Coupler initilize an MCT *Router* between them.
6. Model sends its grid data (lat, lon, area values) to Coupler.

REMARKS:

```
This routine is called by cpl_interface_contractInit.
```

REVISION HISTORY:

```
2002-Jul-30 - T.Craig -- prototype
```

INTERFACE:

```
subroutine cpl_contract_init(srtype,contract,my_name,other_name,buf,decomp)
```

INPUT/OUTPUT PARAMETERS:

```
    character(*)        ,intent(in) :: srtype        ! 'send' or 'recv'
    character(*)        ,intent(in) :: my_name       ! component name (me)
    character(*)        ,intent(in) :: other_name    ! component name (other)
    type(cpl_contract)  ,intent(out):: contract      ! contract
    real(R8),optional   ,intent(in) ::  buf(:,:)     ! data buffer
    integer(IN),optional,intent(in) :: decomp        ! recv side decomp type
                                                     ! 1 = 1d in lat
                                                     ! 2 = 1d in lon
```

4.1.5 cpl_contract_initSend – Initialize contract, send side

Initialize the output `contract` between model `my_name` and model `other_name`. Send `other_name` the grid information in `buf`.
The first dimension of the array `buf` is the local size of the grid. The second dimenstion is `cpl_fields_grid_total`. The contents of the second dimension are the `cpl_fields_grid_fields`.
This calls `cpl_contract_init` with `srtype` equal to "send".

REVISION HISTORY:

```
    2002-Sep-10 - T.Craig -- prototype
```

INTERFACE:

```
subroutine cpl_contract_initSend(contract,my_name,other_name,buf)
```

INPUT/OUTPUT PARAMETERS:

```
    character(*)      ,intent(in)  :: my_name       ! component name (me)
    character(*)      ,intent(in)  :: other_name    ! component name (other)
    type(cpl_contract),intent(out) :: contract      ! contract
    real(R8)          ,intent(in)  ::  buf(:,:)     ! data buffer
```

4.1.6 cpl_contract_initRecv – Initialize contract, receive side

Initialize the output `contract` between model `my_name` and model `other_name`. Recv grid information from `other_name` and store it in the *domain* of the `contract`.
This calls `cpl_contract_init` with `srtype` equal to "recv".

REVISION HISTORY:

```
    2002-Sep-10 - T.Craig -- prototype
```

INTERFACE:

```
subroutine cpl_contract_initRecv(contract,my_name,other_name)
```

INPUT/OUTPUT PARAMETERS:

```
    character(*)      ,intent(in) :: my_name       ! component name (me)
    character(*)      ,intent(in) :: other_name    ! component name (other)
    type(cpl_contract),intent(out):: contract      ! contract
```

5 Interface

5.1 Module cpl_interface_mod – General model-coupler interaction. (Source File: cpl_interface_mod.F90)

This module represents a major subsystem of cpl6.
This module contains the highest level routines for communication between models and the Coupler.
These routines present component models with an API using mostly native Fortran 90 datatypes such as real an integer scalars and arrays. Only one derived datatype, the *contract* appears in the arguments. This satisifes a Coupler requirement to present a simple interface to coupled model programmers. Most of the routines in this module are wrappers to other cpl6 subroutines.
These routines are all that is necessary for a component model to connect to, and exchange data with, version 6 of the CCSM Coupler.

REVISION HISTORY:

```
2003-Jan-15 - T. Craig - change ibuf to infobuf datatype module
2002-Dec-05 - T. Craig - changed call from cpl_coupling to cpl_contract
2002-Sep-10 - T. Craig - abstracted functionality into cpl_coupling_mod.F90
2001-Aug-16 - B. Kauffman - reorganized code according to arch document.
2001-Mar-20 - T. Craig - first prototype
```

INTERFACE:

```
module cpl_interface_mod
```

USES:

```
use cpl_mct_mod         ! mct interface
use cpl_comm_mod        ! mpi/mph communicator info
use cpl_fields_mod      ! coupler/model data field indicies
use cpl_bundle_mod      ! defines bundle
use cpl_domain_mod      ! defines domain
use cpl_infobuf_mod     ! defines infobuf
use cpl_contract_mod    ! defines contract
use cpl_kind_mod        ! defines cpl kinds
use cpl_control_mod, only: dbug=>cpl_control_infoDBug
use shr_sys_mod         ! share system routines
use shr_timer_mod       ! share timer routines
use shr_mpi_mod         ! mpi layer

implicit none

private   ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_interface_init
public :: cpl_interface_finalize
public :: cpl_interface_ibufSend
public :: cpl_interface_ibufRecv
public :: cpl_interface_infobufSend
public :: cpl_interface_infobufRecv
public :: cpl_interface_contractSend
public :: cpl_interface_contractRecv
```

```
public :: cpl_interface_contractInit
public :: cpl_interface_dbugSet   ! set this module's internal dbug level

interface cpl_interface_ibufSend; module procedure cpl_interface_infobufSend; end interface
interface cpl_interface_ibufRecv; module procedure cpl_interface_infobufRecv; end interface
```

PUBLIC DATA MEMBERS:

```
! none
```

5.1.1 cpl_interface_init – initialize the coupling/mpi environment.

Wrapper routine for `cpl_comm_init`. Calls *MPI_Init* and reports model `name` to the coupled system. Returns an *MPI_Communicator* `comm` for use in the calling model.
`name` must be one of the component names in `cpl_fields_mod`.

REVISION HISTORY:

```
2001-Mar-20 - T. Craig, B. Kauffman, R. Jacob -- first prototype
```

INTERFACE:

```
subroutine cpl_interface_init(name,comm)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)  :: name ! name of component name
integer(IN) ,intent(out) :: comm ! communicator group for component
```

5.1.2 cpl_interface_finalize – terminate the coupling/mpi environment.

Calls *MPI_Finalize()* and disengages the model `cname` from the CCSM. `cname` must be one of the component names in `cpl_fields_mod`.

REVISION HISTORY:

```
2001-mmm-dd -
```

INTERFACE:

```
subroutine cpl_interface_finalize(cname)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in) :: cname  ! component name
integer(IN)             :: rcode  ! return code
```

5.1.3 cpl_interface_contractInit – Initialize contract

Initialize the `contract` between model `my_name` and model `other_name`. This is a wrapper to `cpl_contract_init`. Only two sets of optional arguments are allowed. A Model uses the "send" form and should provide `buf`, which contains grid information such as latitude and longitude values (see `cpl_contract_initSend`). The Coupler uses the "recv" form and does not provide `buf`. `bunname` and `fields` are used to initialize the *bundle* in the output `contract`.

REVISION HISTORY:

```
2002-Jul-30 - T.Craig -- prototype
```

INTERFACE:

```
subroutine cpl_interface_contractInit(contract,my_name,other_name,fields,ibufi,buf,ibufr,bunname,dec
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_contract)   ,intent(out)   :: contract   ! contract
character(*)         ,intent(in)    :: my_name    ! my component name
character(*)         ,intent(in)    :: other_name ! other component name
character(*)         ,intent(in)    :: fields     ! fields char string for bun
integer(IN) ,optional,intent(inout) :: ibufi(:)   ! info buffer ints
real(R8)    ,optional,intent(in)    :: buf(:,:)   ! data buffer
real(R8)    ,optional,intent(inout) :: ibufr(:)   ! info buffer reals
character(*),optional,intent(in)    :: bunname
integer(IN) ,optional,intent(in)    :: decomp     ! decomposition type
```

5.1.4 cpl_interface_infobufSend – send an infobuffer using arrays

Load an *infobuffer* using the input arrays `ibufi` and `ibufr` and send it to the root processor of the model with name `cname`.

REVISION HISTORY:

```
2001-Aug-16 -
```

INTERFACE:

```
subroutine cpl_interface_infobufSend(cname,ibufi,ibufr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*)        ,intent(in)       :: cname      ! component name
integer(IN),optional,intent(inout)    :: ibufi(:)   ! info buffer ints
real(R8)   ,optional,intent(inout)    :: ibufr(:)   ! info buffer reals
```

5.1.5 cpl_interface_infobufRecv – receive an infobuffer.

Receive an *infobuffer* from the model with name `cname` and return its integer contents in the `ibufi` array and the real contents in the `ibufr` array.

REVISION HISTORY:

2001-mmm-dd -

INTERFACE:

```
subroutine cpl_interface_infobufRecv(cname,ibufi,ibufr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*)        ,intent(in)  :: cname  ! component name
integer(IN),optional,intent(out) :: ibufi(cpl_infobuf_ibufSize) ! info-buffer
real(R8)   ,optional,intent(out) :: ibufr(cpl_infobuf_rbufSize) ! info-buffer
```

5.1.6 cpl_interface_contractSend – send information in a contract.

Send data in the *infobuffer* and *bundle* of the input `contract` to the component `cname` (which could be a model or the Coupler). If optional arguments `ibufi` and/or `ibufr` are present, that information will be placed in the `contract`'s *infobuffer* and sent instead. If optional argument `buf` is present, then that data will be placed in in the `contract`'s *bundle* and sent to `cname`.
In CCSM3, the Coupler calls this with only `cname` and `contract` while Models use the simple `ibufi` and `buf` arguments.

REVISION HISTORY:

2001-mmm-dd -

INTERFACE:

```
subroutine cpl_interface_contractSend(cname,contract,ibufi,buf,ibufr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*)        ,intent(in)    :: cname      ! component name
type(cpl_contract)  ,intent(inout) :: contract   ! data buffer and domain
integer(IN),optional,intent(inout) :: ibufi(:)   ! info buffer ints
real(R8)   ,optional,intent(inout) :: ibufr(:)   ! info buffer reals
real(R8)   ,optional,intent(inout) ::  buf(:,:)  ! data buffer
```

5.1.7 cpl_interface_contractRecv – Receive information in a contract.

Recive data into the *infobuffer* and *bundle* of the input/output `contract` from the component `cname` (which could be a model or the Coupler). If optional arguments `ibufi` and/or `ibufr` are present, that information is copied out of the `contract`'s *infobuffer* and placed into those arrays. If optional argument `buf` is present, then data in the `contract`'s *bundle* after the receive will be returned in `buf`.
In CCSM3, the Coupler calls this with only `cname` and `contract` while Models use the simple `ibufi` and `buf` arguments.

REVISION HISTORY:

```
2001-mmm-dd -
```

INTERFACE:

```
subroutine cpl_interface_contractRecv(cname,contract,ibufi,buf,ibufr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*)        ,intent(in)    :: cname      ! component name
type(cpl_contract)  ,intent(inout) :: contract   ! data buffer and domain
integer(IN),optional,intent(out)   :: ibufi(:)   ! info buffer ints
real(R8)   ,optional,intent(out)   :: ibufr(:)   ! info buffer reals
real(R8)   ,optional,intent(out)   ::  buf(:,:)  ! data buffer
```

5.1.8 cpl_interface_dbugSet – set this module's internal debug level.

Set this module's internal debug level: 0,1,2,3 (lowest to highest). If debug level is 2 or greater, each call to `cpl_interface_contractSend` or `cpl_interface_contractRecv` will output the global sum of all data sent/received to stdout using `cpl_bundle_gsum`.

REVISION HISTORY:

```
2003-Jan-21 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_interface_dbugSet(level)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN),intent(in) :: level  ! requested debug level
```

6 Fields

6.1 Module cpl_fields_mod – coupler/component list of exchanged fields (Source File: cpl_fields_mod.F90)

This module is an important component of the cpl6 Coupler. It contains the following:

- Master list of all fields exchanged between the Coupler and the component models of CCSM. The fields are listed in a colon-delimited string, e.g. "So_t:So_u:So_v:So_s:So_dhdx:So_dhdy:Fioo_q". The strings below are broken up into one element per line for clarity. Each item in the string becomes an attribute in a *bundle*.
- List of components recogized by the Coupler under the category "component names".
- Integer parameters used by the component models to size and access the simple arrays used in the `cpl_interface_*` routines (`buf, ibufi, etc`). For the field string, there's one integer parameter for each field name with a value equal to its position in the string.

REVISION HISTORY:

```
2002 Feb 11 - full, realistic list of fields
2001 Apr 13 - T. Craig
```

INTERFACE:

```
module cpl_fields_mod
```

USES:

```
use cpl_mct_mod     ! mct
use cpl_kind_mod    ! kinds

private  ! except
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_fields_getField    ! returns string for nth aVect attribute
public :: cpl_fields_getLongName ! returns netCDF longname and unit strings

!-------------------------------------------------------------------------------
! component names
!-------------------------------------------------------------------------------

character(32),parameter,public :: cpl_fields_atmname='atm'
character(32),parameter,public :: cpl_fields_ocnname='ocn'
character(32),parameter,public :: cpl_fields_icename='ice'
character(32),parameter,public :: cpl_fields_lndname='lnd'
character(32),parameter,public :: cpl_fields_rtmname='roff'
character(32),parameter,public :: cpl_fields_cplname='cpl'

!-------------------------------------------------------------------------------
! "info-buffer" index of entries
!-------------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_ibuf_total    = 100 ! size of info-buffer

integer(IN),parameter,public :: cpl_fields_ibuf_rcode    =   1 ! error code
integer(IN),parameter,public :: cpl_fields_ibuf_cdate    =   2 ! current date: yymmdd
integer(IN),parameter,public :: cpl_fields_ibuf_sec      =   3 ! elapsed sec on date
```

```
integer(IN),parameter,public :: cpl_fields_ibuf_ncpl     =   4 ! cpl comm's per day
integer(IN),parameter,public :: cpl_fields_ibuf_nfields  =  10
integer(IN),parameter,public :: cpl_fields_ibuf_gsize    =  11
integer(IN),parameter,public :: cpl_fields_ibuf_lsize    =  12
integer(IN),parameter,public :: cpl_fields_ibuf_gisize   =  13
integer(IN),parameter,public :: cpl_fields_ibuf_gjsize   =  14
integer(IN),parameter,public :: cpl_fields_ibuf_lisize   =  15
integer(IN),parameter,public :: cpl_fields_ibuf_ljsize   =  16
integer(IN),parameter,public :: cpl_fields_ibuf_stopeod  =  19
integer(IN),parameter,public :: cpl_fields_ibuf_stopnow  =  20
integer(IN),parameter,public :: cpl_fields_ibuf_resteod  =  21
integer(IN),parameter,public :: cpl_fields_ibuf_restnow  =  22
integer(IN),parameter,public :: cpl_fields_ibuf_histeod  =  23
integer(IN),parameter,public :: cpl_fields_ibuf_histnow  =  24
integer(IN),parameter,public :: cpl_fields_ibuf_histavg  =  25
integer(IN),parameter,public :: cpl_fields_ibuf_diageod  =  26
integer(IN),parameter,public :: cpl_fields_ibuf_diagnow  =  27
integer(IN),parameter,public :: cpl_fields_ibuf_infotim  =  28
integer(IN),parameter,public :: cpl_fields_ibuf_infobug  =  29
integer(IN),parameter,public :: cpl_fields_ibuf_precadj  =  31 ! precip adjustment factor (* 1.0e
integer(IN),parameter,public :: cpl_fields_ibuf_ashift   =  32 ! albedo calculation time shift
integer(IN),parameter,public :: cpl_fields_ibuf_nbasins  =  33 ! number of active runoff basins
integer(IN),parameter,public :: cpl_fields_ibuf_xalbic   =  34 ! request extra albedo solar init
integer(IN),parameter,public :: cpl_fields_ibuf_inimask  =  36 ! flag cpl to send back domain spe
integer(IN),parameter,public :: cpl_fields_ibuf_dead     =  37 ! non-0 <=> dead model
integer(IN),parameter,public :: cpl_fields_ibuf_domain   =  40
integer(IN),parameter,public :: cpl_fields_ibuf_userest  =  41 ! non-0 <=> use restart data sent

integer(IN),parameter,public :: cpl_fields_rbuf_total    =  50 ! size of real info-buffer

integer(IN),parameter,public :: cpl_fields_rbuf_spval    =   1 ! the special value
integer(IN),parameter,public :: cpl_fields_rbuf_eccen    =  10 ! Earth's eccentricity
integer(IN),parameter,public :: cpl_fields_rbuf_obliqr   =  11 ! Earth's Obliquity
integer(IN),parameter,public :: cpl_fields_rbuf_lambm0   =  12 ! longitude of perihelion at v-equ
integer(IN),parameter,public :: cpl_fields_rbuf_mvelpp   =  13 ! Earth's Moving vernal equinox of

!-------------------------------------------------------------------------
! initial fields, generally a domain description
!-------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_grid_total  = 7

character(*),parameter,public :: cpl_fields_grid_fields = &
   &'lat&
   &:lon&
   &:area&
   &:aream&
   &:index&
   &:mask&
   &:pid'

integer(IN),parameter,public :: cpl_fields_grid_lat    = 1    ! lat from component
integer(IN),parameter,public :: cpl_fields_grid_lon    = 2    ! lon from component
integer(IN),parameter,public :: cpl_fields_grid_area   = 3    ! area from component
integer(IN),parameter,public :: cpl_fields_grid_aream  = 4    ! area from mapping file
integer(IN),parameter,public :: cpl_fields_grid_index  = 5    ! global index
integer(IN),parameter,public :: cpl_fields_grid_mask   = 6    ! mask, 0 = inactive cell
integer(IN),parameter,public :: cpl_fields_grid_pid    = 7    ! proc id number
```

```
!-------------------------------------------------------------------------
! atm fields
!-------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_a2c_total  = 19

character(*), parameter,public :: cpl_fields_a2c_states = &
   &'Sa_z&
   &:Sa_u&
   &:Sa_v&
   &:Sa_tbot&
   &:Sa_ptem&
   &:Sa_shum&
   &:Sa_dens&
   &:Sa_pbot&
   &:Sa_pslv'
character(*), parameter,public :: cpl_fields_a2c_fluxes = &
   &'Faxa_lwdn&
   &:Faxa_rainc&
   &:Faxa_rainl&
   &:Faxa_snowc&
   &:Faxa_snowl&
   &:Faxa_swndr&
   &:Faxa_swvdr&
   &:Faxa_swndf&
   &:Faxa_swvdf&
   &:Faxa_swnet'
character(*), parameter,public :: cpl_fields_a2c_fields = &
   trim(cpl_fields_a2c_states)//":"//trim(cpl_fields_a2c_fluxes)

!----- state fields -----
integer(IN),parameter,public :: cpl_fields_a2c_z      =  1 ! bottom atm level height
integer(IN),parameter,public :: cpl_fields_a2c_u      =  2 ! bottom atm level zon wind
integer(IN),parameter,public :: cpl_fields_a2c_v      =  3 ! bottom atm level mer wind
integer(IN),parameter,public :: cpl_fields_a2c_tbot   =  4 ! bottom atm level temp
integer(IN),parameter,public :: cpl_fields_a2c_ptem   =  5 ! bottom atm level pot temp
integer(IN),parameter,public :: cpl_fields_a2c_shum   =  6 ! bottom atm level spec hum
integer(IN),parameter,public :: cpl_fields_a2c_dens   =  7 ! bottom atm level air den
integer(IN),parameter,public :: cpl_fields_a2c_pbot   =  8 ! bottom atm level pressure
integer(IN),parameter,public :: cpl_fields_a2c_pslv   =  9 ! sea level atm pressure
!----- fluxes computed by atm ----
integer(IN),parameter,public :: cpl_fields_a2c_lwdn   = 10 ! downward lw heat flux
integer(IN),parameter,public :: cpl_fields_a2c_rainc  = 11 ! prec: liquid "convective"
integer(IN),parameter,public :: cpl_fields_a2c_rainl  = 12 ! prec: liquid "large scale"
integer(IN),parameter,public :: cpl_fields_a2c_snowc  = 13 ! prec: frozen "convective"
integer(IN),parameter,public :: cpl_fields_a2c_snowl  = 14 ! prec: frozen "large scale"
integer(IN),parameter,public :: cpl_fields_a2c_swndr  = 15 ! sw: nir direct  downward
integer(IN),parameter,public :: cpl_fields_a2c_swvdr  = 16 ! sw: vis direct  downward
integer(IN),parameter,public :: cpl_fields_a2c_swndf  = 17 ! sw: nir diffuse downward
integer(IN),parameter,public :: cpl_fields_a2c_swvdf  = 18 ! sw: vis diffuse downward
integer(IN),parameter,public :: cpl_fields_a2c_swnet  = 19 ! sw: net


integer(IN),parameter,public :: cpl_fields_c2a_total  = 17

character(*), parameter,public :: cpl_fields_c2a_states = &
   &'Sx_tref&
   &:Sx_qref&
   &:Sx_avsdr&
```

```
   &:Sx_anidr&
   &:Sx_avsdf&
   &:Sx_anidf&
   &:Sx_t&
   &:So_t&
   &:Sx_snowh&
   &:Sx_ifrac&
   &:Sx_ofrac'
character(*), parameter,public :: cpl_fields_c2a_fluxes = &
   &'Faxx_taux&
   &:Faxx_tauy&
   &:Faxx_lat&
   &:Faxx_sen&
   &:Faxx_lwup&
   &:Faxx_evap'
character(*), parameter,public :: cpl_fields_c2a_fields = &
   trim(cpl_fields_c2a_states)//":"//trim(cpl_fields_c2a_fluxes)

!----- states given to atm ----
integer(IN),parameter,public :: cpl_fields_c2a_tref  =  1 ! 2m reference temperature
integer(IN),parameter,public :: cpl_fields_c2a_qref  =  2 ! 2m reference specific humidity
integer(IN),parameter,public :: cpl_fields_c2a_avsdr =  3 ! albedo, visible, direct
integer(IN),parameter,public :: cpl_fields_c2a_anidr =  4 ! albedo, near-ir, direct
integer(IN),parameter,public :: cpl_fields_c2a_avsdf =  5 ! albedo, visible, diffuse
integer(IN),parameter,public :: cpl_fields_c2a_anidf =  6 ! albedo, near-ir, diffuse
integer(IN),parameter,public :: cpl_fields_c2a_t     =  7 ! surface temperature
integer(IN),parameter,public :: cpl_fields_c2a_sst   =  8 ! sea surface temperature
integer(IN),parameter,public :: cpl_fields_c2a_snowh =  9 ! surface snow depth
integer(IN),parameter,public :: cpl_fields_c2a_ifrac = 10 ! surface ice fraction
integer(IN),parameter,public :: cpl_fields_c2a_ofrac = 11 ! surface ocn fraction
!----- fluxes given to atm ----
integer(IN),parameter,public :: cpl_fields_c2a_taux  = 12 ! wind stress, zonal
integer(IN),parameter,public :: cpl_fields_c2a_tauy  = 13 ! wind stress, meridional
integer(IN),parameter,public :: cpl_fields_c2a_lat   = 14 ! latent          heat flux
integer(IN),parameter,public :: cpl_fields_c2a_sen   = 15 ! sensible        heat flux
integer(IN),parameter,public :: cpl_fields_c2a_lwup  = 16 ! upward longwave heat flux
integer(IN),parameter,public :: cpl_fields_c2a_evap  = 17 ! evaporation    water flux

!-------------------------------------------------------------------------
! ice fields
!-------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_i2c_total  = 22

character(*), parameter,public :: cpl_fields_i2c_states = &
   &'Si_t&
   &:Si_tref&
   &:Si_qref&
   &:Si_ifrac&
   &:Si_avsdr&
   &:Si_anidr&
   &:Si_avsdf&
   &:Si_anidf&
   &:index'
character(*), parameter,public :: cpl_fields_i2c_fluxes = &
   &'Faii_taux&
   &:Faii_tauy&
   &:Faii_lat&
   &:Faii_sen&
```

```
   &:Faii_lwup&
   &:Faii_evap&
   &:Faii_swnet&
   &:Fioi_swpen&
   &:Fioi_melth&
   &:Fioi_meltw&
   &:Fioi_salt&
   &:Fioi_taux&
   &:Fioi_tauy'
character(*), parameter,public :: cpl_fields_i2c_fields = &
   trim(cpl_fields_i2c_states)//":"//trim(cpl_fields_i2c_fluxes)

!----- ice states -----
integer(IN),parameter,public :: cpl_fields_i2c_t     =  1 ! temperature
integer(IN),parameter,public :: cpl_fields_i2c_tref  =  2 ! 2m reference temperature
integer(IN),parameter,public :: cpl_fields_i2c_qref  =  3 ! 2m reference specific humidity
integer(IN),parameter,public :: cpl_fields_i2c_ifrac =  4 ! fractional ice coverage
integer(IN),parameter,public :: cpl_fields_i2c_avsdr =  5 ! albedo: visible, direct
integer(IN),parameter,public :: cpl_fields_i2c_anidr =  6 ! albedo: near ir, direct
integer(IN),parameter,public :: cpl_fields_i2c_avsdf =  7 ! albedo: visible, diffuse
integer(IN),parameter,public :: cpl_fields_i2c_anidf =  8 ! albedo: near ir, diffuse
!----- compression index -----
integer(IN),parameter,public :: cpl_fields_i2c_index =  9 ! global data compr index
!----- a/i fluxes computed by ice -----
integer(IN),parameter,public :: cpl_fields_i2c_taux  = 10 ! wind stress, zonal
integer(IN),parameter,public :: cpl_fields_i2c_tauy  = 11 ! wind stress, meridional
integer(IN),parameter,public :: cpl_fields_i2c_lat   = 12 ! latent          heat flux
integer(IN),parameter,public :: cpl_fields_i2c_sen   = 13 ! sensible        heat flux
integer(IN),parameter,public :: cpl_fields_i2c_lwup  = 14 ! upward longwave heat flux
integer(IN),parameter,public :: cpl_fields_i2c_evap  = 15 ! evaporation    water flux
integer(IN),parameter,public :: cpl_fields_i2c_swnet = 16 ! shortwave: net absorbed
!----- i/o fluxes computed by ice -----
integer(IN),parameter,public :: cpl_fields_i2c_swpen = 17 ! net SW penetrating ice
integer(IN),parameter,public :: cpl_fields_i2c_melth = 18 ! heat  flux from melting ice
integer(IN),parameter,public :: cpl_fields_i2c_meltw = 19 ! water flux from melting ice
integer(IN),parameter,public :: cpl_fields_i2c_salt  = 20 ! salt  flux from melting ice
integer(IN),parameter,public :: cpl_fields_i2c_otaux = 21 ! ice/ocn stress, zonal
integer(IN),parameter,public :: cpl_fields_i2c_otauy = 22 ! ice/ocn stress, meridional


integer(IN),parameter,public :: cpl_fields_c2i_total  = 21

character(*), parameter,public :: cpl_fields_c2i_states = &
   &'So_t&
   &:So_s&
   &:So_u&
   &:So_v&
   &:Sa_z&
   &:Sa_u&
   &:Sa_v&
   &:Sa_ptem&
   &:Sa_tbot&
   &:Sa_shum&
   &:Sa_dens&
   &:So_dhdx&
   &:So_dhdy'
character(*), parameter,public :: cpl_fields_c2i_fluxes = &
   &'Fioo_q&
   &:Faxa_swndr&
```

```
   &:Faxa_swvdr&
   &:Faxa_swndf&
   &:Faxa_swvdf&
   &:Faxa_lwdn&
   &:Faxc_rain&
   &:Faxc_snow'
character(*), parameter,public :: cpl_fields_c2i_fields = &
   trim(cpl_fields_c2i_states)//":"//trim(cpl_fields_c2i_fluxes)

!----- ocn states -----
integer(IN),parameter,public :: cpl_fields_c2i_ot    =  1 ! ocn temp
integer(IN),parameter,public :: cpl_fields_c2i_os    =  2 ! ocn salinity
integer(IN),parameter,public :: cpl_fields_c2i_ou    =  3 ! ocn u velocity
integer(IN),parameter,public :: cpl_fields_c2i_ov    =  4 ! ocn v velocity
integer(IN),parameter,public :: cpl_fields_c2i_dhdx  = 12 ! ocn surface slope, zonal
integer(IN),parameter,public :: cpl_fields_c2i_dhdy  = 13 ! ocn surface slope, merid
!----- atm states -----
integer(IN),parameter,public :: cpl_fields_c2i_z     =  5 ! atm bottom layer height
integer(IN),parameter,public :: cpl_fields_c2i_u     =  6 ! atm u velocity
integer(IN),parameter,public :: cpl_fields_c2i_v     =  7 ! atm v velocity
integer(IN),parameter,public :: cpl_fields_c2i_ptem  =  8 ! atm potential temp
integer(IN),parameter,public :: cpl_fields_c2i_tbot  =  9 ! atm bottom temp
integer(IN),parameter,public :: cpl_fields_c2i_shum  = 10 ! atm specfic humidity
integer(IN),parameter,public :: cpl_fields_c2i_dens  = 11 ! atm air density
!----- ocn fluxes -----
integer(IN),parameter,public :: cpl_fields_c2i_q     = 14 ! ocn freeze or melt heat
!----- atm fluxes -----
integer(IN),parameter,public :: cpl_fields_c2i_swndr = 15 ! atm sw near-ir, direct
integer(IN),parameter,public :: cpl_fields_c2i_swvdr = 16 ! atm sw visable, direct
integer(IN),parameter,public :: cpl_fields_c2i_swndf = 17 ! atm sw near-ir, diffuse
integer(IN),parameter,public :: cpl_fields_c2i_swvdf = 18 ! atm sw visable, diffuse
integer(IN),parameter,public :: cpl_fields_c2i_lwdn  = 19 ! long-wave down
integer(IN),parameter,public :: cpl_fields_c2i_rain  = 20 ! rain
integer(IN),parameter,public :: cpl_fields_c2i_snow  = 21 ! snow

!-------------------------------------------------------------------------------
! lnd fields
!-------------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_l2c_total  = 15

character(*), parameter,public :: cpl_fields_l2c_states = &
   &'Sl_t&
   &:Sl_tref&
   &:Sl_qref&
   &:Sl_avsdr&
   &:Sl_anidr&
   &:Sl_avsdf&
   &:Sl_anidf&
   &:Sl_snowh'
character(*), parameter,public :: cpl_fields_l2c_fluxes = &
   &'Fall_taux&
   &:Fall_tauy&
   &:Fall_lat&
   &:Fall_sen&
   &:Fall_lwup&
   &:Fall_evap&
   &:Fall_swnet'
character(*), parameter,public :: cpl_fields_l2c_fields = &
```

```
   trim(cpl_fields_l2c_states)//":"//trim(cpl_fields_l2c_fluxes)

!----- lnd states -----
integer(IN),parameter,public :: cpl_fields_l2c_t    =  1 ! temperature
integer(IN),parameter,public :: cpl_fields_l2c_tref  =  2 ! 2m reference temperature
integer(IN),parameter,public :: cpl_fields_l2c_qref  =  3 ! 2m reference specific humidity
integer(IN),parameter,public :: cpl_fields_l2c_avsdr =  4 ! albedo: direct , visible
integer(IN),parameter,public :: cpl_fields_l2c_anidr =  5 ! albedo: direct , near-ir
integer(IN),parameter,public :: cpl_fields_l2c_avsdf =  6 ! albedo: diffuse, visible
integer(IN),parameter,public :: cpl_fields_l2c_anidf =  7 ! albedo: diffuse, near-ir
integer(IN),parameter,public :: cpl_fields_l2c_snowh =  8 ! snow height
!----- computed by lnd -----
integer(IN),parameter,public :: cpl_fields_l2c_taux  =  9 ! wind stress, zonal
integer(IN),parameter,public :: cpl_fields_l2c_tauy  = 10 ! wind stress, meridional
integer(IN),parameter,public :: cpl_fields_l2c_lat   = 11 ! latent          heat flux
integer(IN),parameter,public :: cpl_fields_l2c_sen   = 12 ! sensible        heat flux
integer(IN),parameter,public :: cpl_fields_l2c_lwup  = 13 ! upward longwave heat flux
integer(IN),parameter,public :: cpl_fields_l2c_evap  = 14 ! evaporation    water flux
integer(IN),parameter,public :: cpl_fields_l2c_swnet = 15 ! 2m reference temperature


integer(IN),parameter,public :: cpl_fields_c2l_total  = 18

character(*), parameter,public :: cpl_fields_c2l_states = &
   &'Sa_z&
   &:Sa_u&
   &:Sa_v&
   &:Sa_tbot&
   &:Sa_ptem&
   &:Sa_shum&
   &:Sa_dens&
   &:Sa_pbot&
   &:Sa_pslv'
character(*), parameter,public :: cpl_fields_c2l_fluxes = &
   &'Faxa_lwdn&
   &:Faxa_rainc&
   &:Faxa_rainl&
   &:Faxa_snowc&
   &:Faxa_snowl&
   &:Faxa_swndr&
   &:Faxa_swvdr&
   &:Faxa_swndf&
   &:Faxa_swvdf'
character(*), parameter,public :: cpl_fields_c2l_fields = &
   trim(cpl_fields_c2l_states)//":"//trim(cpl_fields_c2l_fluxes)

!----- atm states -----
integer(IN),parameter,public :: cpl_fields_c2l_z    =  1 ! bottom atm level height
integer(IN),parameter,public :: cpl_fields_c2l_u    =  2 ! bottom atm level zon wind
integer(IN),parameter,public :: cpl_fields_c2l_v    =  3 ! bottom atm level mer wind
integer(IN),parameter,public :: cpl_fields_c2l_tbot  =  4 ! bottom atm level temp
integer(IN),parameter,public :: cpl_fields_c2l_ptem  =  5 ! bottom atm level pot temp
integer(IN),parameter,public :: cpl_fields_c2l_shum  =  6 ! bottom atm level spec hum
integer(IN),parameter,public :: cpl_fields_c2l_dens  =  7 ! bottom atm level air dens
integer(IN),parameter,public :: cpl_fields_c2l_pbot  =  8 ! bottom atm level pressure
integer(IN),parameter,public :: cpl_fields_c2l_pslv  =  9 ! sea level atm pressure
!----- computed by atm -----
integer(IN),parameter,public :: cpl_fields_c2l_lwdn  = 10 ! downward longwave heat flux
integer(IN),parameter,public :: cpl_fields_c2l_rainc = 11 ! precip: liquid, convective
```

```
integer(IN),parameter,public :: cpl_fields_c2l_rainl = 12 ! precip: liquid, large-scale
integer(IN),parameter,public :: cpl_fields_c2l_snowc = 13 ! precip: frozen, convective
integer(IN),parameter,public :: cpl_fields_c2l_snowl = 14 ! precip: frozen, large-scale
integer(IN),parameter,public :: cpl_fields_c2l_swndr = 15 ! shortwave: nir direct  down
integer(IN),parameter,public :: cpl_fields_c2l_swvdr = 16 ! shortwave: vis direct  down
integer(IN),parameter,public :: cpl_fields_c2l_swndf = 17 ! shortwave: nir diffuse down
integer(IN),parameter,public :: cpl_fields_c2l_swvdf = 18 ! shortwave: vis diffuse down

!----- special lnd grid initialization -----

integer(IN),parameter,public :: cpl_fields_c2lg_total  = 6

character(*), parameter,public :: cpl_fields_c2lg_fields = &
   &'lon&
   &:lat&
   &:area&
   &:lfrac&
   &:maskl&
   &:maska'

integer(IN),parameter,public :: cpl_fields_c2lg_alon  =  1 ! longitude
integer(IN),parameter,public :: cpl_fields_c2lg_alat  =  2 ! latitude
integer(IN),parameter,public :: cpl_fields_c2lg_aarea =  3 ! cell area
integer(IN),parameter,public :: cpl_fields_c2lg_lfrac =  4 ! lnd fraction
integer(IN),parameter,public :: cpl_fields_c2lg_lmask =  5 ! lnd mask
integer(IN),parameter,public :: cpl_fields_c2lg_amask =  6 ! atm mask

!-------------------------------------------------------------------------
! ocn fields
!-------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_o2c_total  = 7

character(*), parameter,public :: cpl_fields_o2c_states = &
   &'So_t&
   &:So_u&
   &:So_v&
   &:So_s&
   &:So_dhdx&
   &:So_dhdy'
character(*), parameter,public :: cpl_fields_o2c_fluxes = &
   &'Fioo_q'
character(*), parameter,public :: cpl_fields_o2c_fields = &
   trim(cpl_fields_o2c_states)//":"//trim(cpl_fields_o2c_fluxes)


!----- ocn states -----
integer(IN),parameter,public :: cpl_fields_o2c_t    =  1 ! temperature
integer(IN),parameter,public :: cpl_fields_o2c_u    =  2 ! velocity, zonal
integer(IN),parameter,public :: cpl_fields_o2c_v    =  3 ! velocity, meridional
integer(IN),parameter,public :: cpl_fields_o2c_s    =  4 ! salinity
integer(IN),parameter,public :: cpl_fields_o2c_dhdx =  5 ! surface slope, zonal
integer(IN),parameter,public :: cpl_fields_o2c_dhdy =  6 ! surface slope, meridional
integer(IN),parameter,public :: cpl_fields_o2c_q    =  7 ! heat of fusion (q>0) melt pot (q<0)


integer(IN),parameter,public :: cpl_fields_c2o_total  = 18

character(*), parameter,public :: cpl_fields_c2o_states = &
```

```
   &'Si_ifrac&
   &:Sa_pslv&
   &:Faoc_duu10n'
character(*), parameter,public :: cpl_fields_c2o_fluxes = &
   &'Foxx_taux&
   &:Foxx_tauy&
   &:Foxx_swnet&
   &:Foxx_lat&
   &:Foxx_sen&
   &:Foxx_lwup&
   &:Foxx_lwdn&
   &:Foxx_melth&
   &:Foxx_salt&
   &:Foxx_prec&
   &:Foxx_snow&
   &:Foxx_rain&
   &:Foxx_evap&
   &:Foxx_meltw&
   &:Forr_roff'
character(*), parameter,public :: cpl_fields_c2o_fields = &
   trim(cpl_fields_c2o_states)//":"//trim(cpl_fields_c2o_fluxes)

!----- ocn model input -----
integer(IN),parameter,public :: cpl_fields_c2o_ifrac =  1 ! state: ice fraction
integer(IN),parameter,public :: cpl_fields_c2o_press =  2 ! state: sea level pressure
integer(IN),parameter,public :: cpl_fields_c2o_duu10 =  3 ! state: 10m wind speed squared
integer(IN),parameter,public :: cpl_fields_c2o_taux  =  4 ! wind stress: zonal
integer(IN),parameter,public :: cpl_fields_c2o_tauy  =  5 ! wind stress: meridional
integer(IN),parameter,public :: cpl_fields_c2o_swnet =  6 ! heat flux: shortwave net
integer(IN),parameter,public :: cpl_fields_c2o_lat   =  7 ! heat flux: latent
integer(IN),parameter,public :: cpl_fields_c2o_sen   =  8 ! heat flux: sensible
integer(IN),parameter,public :: cpl_fields_c2o_lwup  =  9 ! heat flux: long-wave up
integer(IN),parameter,public :: cpl_fields_c2o_lwdn  = 10 ! heat flux: long-wave down
integer(IN),parameter,public :: cpl_fields_c2o_melth = 11 ! heat flux: melt
integer(IN),parameter,public :: cpl_fields_c2o_salt  = 12 ! salt flux
integer(IN),parameter,public :: cpl_fields_c2o_prec  = 13 ! water flux: rain+snow
integer(IN),parameter,public :: cpl_fields_c2o_snow  = 14 ! water flux: snow
integer(IN),parameter,public :: cpl_fields_c2o_rain  = 15 ! water flux: rain
integer(IN),parameter,public :: cpl_fields_c2o_evap  = 16 ! water flux: evap
integer(IN),parameter,public :: cpl_fields_c2o_meltw = 17 ! water flux: melt
integer(IN),parameter,public :: cpl_fields_c2o_roff  = 18 ! water flux: runoff

!-------------------------------------------------------------------------
! run-off field
!-------------------------------------------------------------------------

integer(IN),parameter,public :: cpl_fields_r2c_total  = 1

character(*), parameter,public :: cpl_fields_r2c_states = &
   &''
character(*), parameter,public :: cpl_fields_r2c_fluxes = &
   &'Forr_roff'
character(*), parameter,public :: cpl_fields_r2c_fields = &
  trim(cpl_fields_r2c_states)//":"//trim(cpl_fields_r2c_fluxes)
character(*), parameter,public :: cpl_fields_r2c_fields = &
                                    trim(cpl_fields_r2c_fluxes)

integer(IN),parameter,public :: cpl_fields_r2c_runoff = 1
```

6.1.1 cpl_fields_getField

Returns `nfld` element of the colon-delimited string `cstring` in the output character string `outfield`.

REVISION HISTORY:

```
2003-Jan-24  - T. Craig - first version
```

INTERFACE:

```
subroutine cpl_fields_getField(outfield,nfld,cstring)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(out) :: outfield   ! output field name
integer     ,intent(in ) :: nfld       ! field number
character(*),intent(in ) :: cstring    ! colon delimited field string
```

6.2 Module cpl_fields_getLongName – get netCDF attributes for a field (Source File: cpl_fields_mod.F90)

Parse the field name `fldstr` and and return the netCDF attribute character string `longname` and unit string `units` corresponding to the given field name.
Contructs a lookup table of short and long names.
Example: for input `So_dhdx`, the `So_` is removed and the attribute and units for `dydx` is returned.

REVISION HISTORY:

```
2003-may-12 - B. Kauffman - initial version
```

INTERFACE:

```
subroutine cpl_fields_getLongName(fldstr,longname,units)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)   :: fldstr     ! field name
character(*),intent(out)  :: longname   ! corresponding longname
character(*),intent(out)  :: units      ! corresponding units
```

7 Mapping (Interpolation)

7.1 Module cpl_map_mod – mapping subsystem module (Source File: cpl_map_mod.F9

This module represents a major subsystem of cpl6. "Mapping" refers to the interpolation of 2d field data from one domain/grid to another. It is often desirable that maps have the properties of being *smooth* and *conservative*. Common mapping techniques are bilinear interpolation and area-averaging. Mapping in cpl6 is implemented by a sparse matrix multiply. This module defines the `cpl_map` data type which hold all the information needed to complete a mapping of a *bundle* from one *domain* to another. It also includes routines to do the mapping and initialize the `cpl_map` data structure and check the properties of the weights from the sparse matrix.

REVISION HISTORY:

```
2001-Aug-14 - B. Kauffman -- gathered all mapping routines into this module
2001-May-20 - T. Craig -- first prototype
```

INTERFACE:

```
module cpl_map_mod
```

USES:

```
use cpl_mct_mod        ! mct interface
use cpl_domain_mod     ! data type & methods
use cpl_bundle_mod     ! data type & methods
use cpl_comm_mod       ! global data
use cpl_kind_mod       ! kinds
use cpl_control_mod, only: dbug=>cpl_control_infoDBug
use cpl_control_mod, only: bfbflag=>cpl_control_bfbflag
use shr_sys_mod        ! flush
use shr_mpi_mod        ! mpi layer

implicit none

private   ! except
```

PUBLIC TYPES:

```
public :: cpl_map

type cpl_map
  character(CL)            :: name    ! text ID of mapping data
  type(cpl_mct_sMat)       :: sMat    ! the mct sparse matrix data type
  type(cpl_domain),pointer :: src     ! the associated source domain
  type(cpl_domain),pointer :: dst     ! the associated destination domain
  type(cpl_domain)         :: new     ! new/intermediate domain required by mct
  type(cpl_mct_rearr)      :: rearr   ! rearranger to/from new
  character(3)             :: newtype ! intermediate domain type: src or dst ?
  integer(IN)              :: IDtype  ! 0=normal, 1=identity(ID)
  type(cpl_mct_Avect)      :: areasrc ! area of src grid from mapping file
  type(cpl_mct_Avect)      :: areadst ! area of dst grid from mapping file
end type cpl_map
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_map_init      ! initialize a map
public :: cpl_map_clean     ! clean/dealloc a map
public :: cpl_map_info      ! obtain information about a map
public :: cpl_map_bun       ! map from one bundle to another
```

```
public :: cpl_map_npFix     ! fix NP values wrt mapping vector fields

interface cpl_map_npFix; module procedure cpl_map_npFixNew3; end interface
interface cpl_map_npFix; module procedure cpl_map_npFixNew2; end interface
interface cpl_map_npFix; module procedure cpl_map_npFixNew; end interface
interface cpl_map_npFix; module procedure cpl_map_npFixOld; end interface
interface cpl_map_npFix; module procedure cpl_map_npFixNone; end interface
```

PUBLIC DATA MEMBERS:

```
character(*),parameter,public :: cpl_map_areaAV_field = 'aream'
```

7.1.1 cpl_map_init - Create a map between two domains

Initialize a `map_X` to interpolate data from *domain* `dom_src` to *domain* `dom_dst`. `map_X` is assigned the name `mapName`.
Mapping weights are read from the file `fileName`
`newdom` is either "src" or "dst" and specifies if the communication needed to complete the mapping is done before, "src"-based, or after, "dst"-based, the mapping.
If optional argument `adj_areas` is present and true, the mapping weights will be adjusted to account for area differences between the models and the SCRIP weight generation program. Experimental. Do not use.
If `cpl_control_infoDBug` is greater than 1, then this routine will perform consistency checks on the mapping weights.

REVISION HISTORY:

```
2001-Jun-14 - T. Craig - first functioning version
2001-Mar-20 - T. Craig, B. Kauffman, R. Jacob - first prototype
```

INTERFACE:

```
subroutine cpl_map_init(map_X,dom_src,dom_dst,mapName,fileName,newdom,adj_areas)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_map)   ,intent(out)        :: map_X    ! map_X data
type(cpl_domain),intent( in),target :: dom_src  ! map's source domain
type(cpl_domain),intent( in),target :: dom_dst  ! map's destination domain
character(*),intent( in)            :: mapName  ! map's ID string
character(*),intent( in)            :: fileName ! file containing map data
character(*),intent( in)            :: newdom   ! which domain to alter (for mct)
logical,optional                    :: adj_areas! flag to adjust areas
```

7.1.2 cpl_map_clean - Deallocate a map data type

Deallocate all memory associated with the input map type `mapping`.

REVISION HISTORY:

```
2002-Jan-20 - T. Craig - first functioning version
```

INTERFACE:

```
subroutine cpl_map_clean(mapping)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_map),intent(inout)    :: mapping ! mapping data
```

7.1.3 cpl_map_info - Print inforomation about the map

Print information about the map `mapping` to stdout.

REVISION HISTORY:

```
      2002-Jan-14 - T. Craig - first functioning version
```

INTERFACE:

```
subroutine cpl_map_info(mapping)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_map)   ,intent(in)                  :: mapping ! mapping data
```

7.1.4 cpl_map_bun - map a bundle from one domain to domain

Map input bundle `buni` on one *domain* to output bundle `buno` on a different *domain* using the map `mapx`.
All attributes in `buni` and `buno` which have the same name will be mapped. Data in `buno` will be overwritten.
Both `buni` and `buno` must be initialized before calling this routine
If the set of optional arguments `bunfs`, `fsname`, `bunfd`, `fsname` are all present then the data in `buni` will be multiplied by field `fsname` from `bunfs` before mapping (Note: the data in `buni` will NOT be altered) and the data in `buno` will be multipled by field `fdname` in bundle `bunfs` after mapping.
If optional argument `mvector` controls the use of the vector-computer freindly versions of the mapping routine. This can be used to override the default settings.

REVISION HISTORY:

```
      20May01 - T. Craig -- first prototype
      15Jun01 - E.T. Ong -- Removed zeroing of bunn%data and buno%data -
                            this is done in mct calls.
```

INTERFACE:

```
subroutine cpl_map_bun(buni,buno,mapx,bunfs,fsname,bunfd,fdname,mvector)
```

USES:

```
    use shr_timer_mod       ! share timer routines
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout)       :: buni    ! input bundle
type(cpl_bundle),intent(out)         :: buno    ! output bundle
type(cpl_map)   ,intent(inout)       :: mapx    ! mapping between two domains
type(cpl_bundle),intent(in),optional :: bunfs   ! src fraction input bundle
character(*)    ,intent(in),optional :: fsname  ! name of field in bunfs
type(cpl_bundle),intent(in),optional :: bunfd   ! dst fraction input bundle
character(*)    ,intent(in),optional :: fdname  ! name of field in bunfd
logical         ,intent(in),optional :: mvector  ! enable vector-friendly mapping
```

7.1.5 cpl_map_npFixNew3 - correct the north pole mapping of velocity fields

Correct the north pole mapping of velocity fields from the atm to ocn grids. This assumes the input grid is a regular lat/lon with the north pole surrounded by the last latitude line in the input array. The longitudes in the last latitude must be ordered and equally spaced.

REVISION HISTORY:

```
29Aug03 - T. Craig -- first prototype
```

INTERFACE:

```
subroutine cpl_map_npFixNew3(buni,buno,fld1,fld2)
```

USES:

```
use cpl_const_mod
use shr_timer_mod       ! share timer routines

#if (! defined HIDE_MPI)
#include <mpif.h>         ! mpi library include file
#endif
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout):: buni    ! input bundle
type(cpl_bundle),intent(out)  :: buno    ! output bundle
character(*)    ,intent(in)   :: fld1    ! name of first input field
character(*)    ,intent(in)   :: fld2    ! name of second input field
```

8 Namelist and Control Variable

8.1 Module cpl_control_mod – basic coupler control function logic. (Source File: cpl_control_mod.F90)

This module represents a major subsystem of cpl6. It contains data type definitions and associated methods used for controlling a coupled integration. Here "controlling" refers to issues such as:

- selecting integration start and stop dates,
- determining when history and restart data files should be generated,
- determining when diagnostic data should be generated,
- verifying that all coupled system component models (eg. atm, ocn, etc) are synchronized in time.
- reads and parses namelist variables
- makes various simulation control variables available to other modules

REVISION HISTORY:

```
2002-Sep-18 - B. Kauffman -- reworked using shr_alarm_mod
2001-May-27 - T. Bettge -- initial prototype
```

INTERFACE:

```
module cpl_control_mod
```

USES:

```
use shr_sys_mod     ! wrappers around system calls
use shr_cal_mod     ! calendar module
use shr_date_mod    ! date data-type & methods
use shr_alarm_mod   ! alarm data-type & methods
use cpl_kind_mod    ! access to F90 kind declarations

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_control_readNList  ! read & parse namelist input
public :: cpl_control_init       ! initialize alarms, etc.
public :: cpl_control_update     ! update control flags
```

PUBLIC DATA MEMBERS:

```
!----- rest, stop, hist, diag control/alarm flags -----
logical      ,public :: cpl_control_stopNow    ! T => stop model now
logical      ,public :: cpl_control_stopEOD    ! T => stop model at end of day
logical      ,public :: cpl_control_restNow    ! T => create restart data now
logical      ,public :: cpl_control_restEOD    ! T => create restart data at EOD
logical      ,public :: cpl_control_histNow    ! T => create history data now
logical      ,public :: cpl_control_histEOD    ! T => create history data at EOD
```

```
logical      ,public :: cpl_control_histSave   ! T => archive history data now
logical      ,public :: cpl_control_hist64bit  ! T => use 64 bit netCDFfiles
logical      ,public :: cpl_control_avhistNow  ! T => create history data now
logical      ,public :: cpl_control_avhistEOD  ! T => create history data at EOD
logical      ,public :: cpl_control_diagNow    ! T => print diagnostic data now
logical      ,public :: cpl_control_diagEOD    ! T => print diagnostic data at EOD
logical      ,public :: cpl_control_avDiagNow  ! T => print tavg diag data now
logical      ,public :: cpl_control_avDiagEOD  ! T => print tavg diag data at EOD
logical      ,public :: cpl_control_bfbflag    ! T => bfb with different pes

!----- case name & descriptive string -----
character(CL),public :: cpl_control_caseName   ! case name
character(CL),public :: cpl_control_caseDesc   ! case description

!----- restart control -----
character(16),public :: cpl_control_restType   ! restart type: init,cont,branch
integer(IN)  ,public :: cpl_control_restCDate  ! restart cDate from namelist
integer(IN)  ,public :: cpl_control_restDate   ! restart date
character(CL),public :: cpl_control_restPFn    ! restart pointer file name
character(CL),public :: cpl_control_restBFn    ! restart branch  file name
logical      ,public :: cpl_control_lagOcn     ! T => lag the ocn at startup
logical      ,public :: cpl_control_sendAtmAlb ! T => send albedo ICs to atm
logical      ,public :: cpl_control_sendLndDom ! T => send lnd domain to lnd
logical      ,public :: cpl_control_icData_a   ! T => use IC data provided by atm
logical      ,public :: cpl_control_icData_i   ! T => use IC data provided by ice
logical      ,public :: cpl_control_icData_l   ! T => use IC data provided by lnd
logical      ,public :: cpl_control_icData_o   ! T => use IC data provided by ocn
logical      ,public :: cpl_control_icData_r   ! T => use IC data provided by roff
character(16),public :: cpl_control_avhistType ! tavg history file type

!----- mapping file names -----
character(CL),public :: cpl_control_mapFn_a2oF ! map data file: a->o fluxes
character(CL),public :: cpl_control_mapFn_a2oS ! map data file: a->o states
character(CL),public :: cpl_control_mapFn_o2aF ! map data file: o->a fluxes
character(CL),public :: cpl_control_mapFn_o2aS ! map data file: o->a states
character(CL),public :: cpl_control_mapFn_r2o  ! map data file: r->o runoff

!----- flux & orbital options -----
logical      ,public :: cpl_control_fluxAlbAv  ! T => NO diurnal cycle in albedos
character(16),public :: cpl_control_fluxEPbal  ! selects E,P,R adjustment technique
real(R8)     ,public :: cpl_control_fluxEPfac  ! E,P,R adjust factor recv'd from ocn
integer(IN)  ,public :: cpl_control_fluxAShift ! albedo calc time-shift (seconds)
real(R8)     ,public :: cpl_control_orbEccen   ! eccen of earth orbit (unitless)
real(R8)     ,public :: cpl_control_orbObliqr  ! earth's obliquity (rad)
real(R8)     ,public :: cpl_control_orbLambm0  ! mean lon perihelion @ vernal eq (rad)
real(R8)     ,public :: cpl_control_orbMvelpp  ! moving vernal equinox longitude
                                               ! of perihelion plus pi (rad)

!----- info about which specific component models are in use -----
logical      ,public :: cpl_control_dead_a     ! T => atm component is dead comp
logical      ,public :: cpl_control_dead_i     ! T => ice component is dead comp
logical      ,public :: cpl_control_dead_l     ! T => lnd component is dead comp
logical      ,public :: cpl_control_dead_o     ! T => ocn component is dead comp
logical      ,public :: cpl_control_dead_ao    ! T => atm and/or ocn are dead comp

!----- date/time & timestep info -----
integer(IN)  ,public :: cpl_control_nCpl_a     ! atm/cpl communications per day
integer(IN)  ,public :: cpl_control_nCpl_i     ! ice/cpl communications per day
integer(IN)  ,public :: cpl_control_nCpl_l     ! lnd/cpl communications per day
```

```
integer(IN)  ,public :: cpl_control_nCpl_o     ! ocn/cpl communications per day
integer(IN)  ,public :: cpl_control_nCpl_r     ! rof/cpl communications per day
integer(IN)  ,public :: cpl_control_cDate_a    ! atm coded date
integer(IN)  ,public :: cpl_control_cDate_i    ! ice coded date
integer(IN)  ,public :: cpl_control_cDate_l    ! lnd coded date
integer(IN)  ,public :: cpl_control_cDate_o    ! ocn coded date
integer(IN)  ,public :: cpl_control_sec_a      ! atm secs on date
integer(IN)  ,public :: cpl_control_sec_i      ! ice secs on date
integer(IN)  ,public :: cpl_control_sec_l      ! lnd secs on date
integer(IN)  ,public :: cpl_control_sec_o      ! ocn secs on date

!----- decomposition settings -----
integer(IN)  ,public :: cpl_control_decomp_a   ! atm decomposition type
integer(IN)  ,public :: cpl_control_decomp_l   ! lnd decomposition type
integer(IN)  ,public :: cpl_control_decomp_o   ! ocn decomposition type
integer(IN)  ,public :: cpl_control_decomp_i   ! ice decomposition type
integer(IN)  ,public :: cpl_control_decomp_r   ! rof decomposition type

!----- other flags -----
integer(IN)  ,public :: cpl_control_infoDBug=1 ! user specified dbug level
logical      ,public :: cpl_control_infoBcheck ! T => do bit-check now
```

8.1.1 cpl_control_readNList - initialize and read namelist values.

Initialize and read namelist values.

REVISION HISTORY:

```
2002-Sep-18 - B. Kauffman -- 1st version
```

INTERFACE:

```
subroutine cpl_control_readNList()
```

USES:

```
use shr_orb_mod

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
! modifies private module data for later use (the namelist variables)
```

8.1.2 cpl_control_init - initializes flags for stopping, restart, etc.

Set the module variable `startDate` to the input argument `date`. Also set the module variables `stopAlarm`, `restAlarm`, `histAlarm`, `avhistAlarm`, `diagAlarm` and `avdiagAlarm`. If any of `rest_date, stop_date, etc.` are unset or negative, set them to the input `date`.

REVISION HISTORY:

```
2002-Sep-18 - B. Kauffman -- 1st version
```

INTERFACE:

```
subroutine cpl_control_init(date)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
!output: modifies data declared in this module's header
type(shr_date),intent(in) :: date ! model start date
```

8.1.3 cpl_control_update - sets control flags for stopping,restart,etc.

Update all the module `cpl_control_*Now` and `cpl_control_*EOD` flags using input argument `currentDate`. Also set `cpl_control_infoBCheck`.

REVISION HISTORY:

```
2002-Sep-18 - B. Kauffman -- 1st version using shr_alarm_mod
```

INTERFACE:

```
subroutine cpl_control_update(currentDate)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(shr_date),intent(in) :: currentDate  ! current model date

!OUTPUT: modifies public cpl_control_mod data
```

9 Binary IO

9.1 Module cpl_iobin_mod – create, write-to, or read a binary data file. (Source File: cpl_iobin_mod.F90)

This module creates, writes-to, and reads data in a *bundle* to/from a file in machine-dependent binary format. This module is intended to support the restart sub-system of cpl6.

REMARKS:

```
This module could be used to create binary data files for any
purpose.  The binary data file format is self-describing and extensible and
thus might provide an alternative to netCDF files.
```

REVISION HISTORY:

```
 2002-nov-08 - B. Kauffman - initial version
```

INTERFACE:

```
module cpl_iobin_mod
```

USES:

```
use cpl_mct_mod       ! mct interface
use cpl_comm_mod      ! mpi/mph communicator info
use cpl_fields_mod    ! coupler/model data field indices
use cpl_bundle_mod    ! defines bundle
use cpl_domain_mod    ! defines domain
use cpl_kind_mod      ! defines F90 kinds
use cpl_control_mod, only: dbug=>cpl_control_infoDBug
use shr_sys_mod       ! share system routines
use shr_date_mod      ! defines date data-type
use shr_mpi_mod       ! layer on MPI

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_iobin_create     ! create a new file (an empty file)
public :: cpl_iobin_open       ! open a named file
public :: cpl_iobin_close      ! close an open file
public :: cpl_iobin_appendBun  ! add  bundle data to   a file
public :: cpl_iobin_readBun    ! read bundle data from a file
public :: cpl_iobin_appendReal ! add  real array data   to a file
public :: cpl_iobin_readReal   ! read real array data from a file
public :: cpl_iobin_readDate   ! read data date from a file
```

PUBLIC DATA MEMBERS:

```
! none
```

9.1.1 cpl_iobin_create – create a new file.

Open a file with name `fName` and write a small header of 6 character strings each with length CL. Use Fortran unformatted write. If optional argument `desc` is present, it will be included in the header.

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_create(fName,desc)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)          :: fName   ! file name
character(*),intent(in),optional :: desc    ! description string
```

9.1.2 cpl_iobin_open – open an existing file.

Open the pre-existing file with name `fName` and assign it the unit number `fid`. Also read the header information and write it to stdout.

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_open(fName,fid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)   :: fName   ! file name
integer(IN) ,intent(out)  :: fid     ! file ID (file unit number)
```

9.1.3 cpl_iobin_close – close an open file.

Call `close` on file with unit number `fid`.

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_close(fid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN),intent(in)  :: fid    ! file ID (file unit number)
```

9.1.4 cpl_iobin_appendBun – add bundle data to an existing file.

Append the data in *bundle* `bun` to the pre-existing file with the unit number `fid`. Also write the date contained in `date` to the file. `fid` must be a valid fortran unit number for an open file.
All processors call this function and the root node will *MPI_gather* the data and write it to the file.

REMARKS:

```
Domain data associated with the bundle is not written
but this functionality could be added, if desired.
The file format utilizes an extensible, self-describing data format.
```

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_appendBun(fid,date,bun)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN)          ,intent(inout) :: fid     ! file ID
type(shr_date)       ,intent(in)    :: date    ! model date
type(cpl_bundle)     ,intent(in)    :: bun     ! bundle
```

9.1.5 cpl_iobin_readBun – read bundle data from a file.

Read data from file with unit number `fid` and return in in the *bundle* `bun`. Argument `date` is currently ignored. Data is read on node 0 and scattered using the information in the *domain* associated with `bun`. On return, `bun` contains the data for points local to the calling processor.

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_readBun(fid,date,bun)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN)     ,intent(in)    :: fid   ! file ID
type(shr_date)  ,intent(inout) :: date  ! desired model date, file's date?
type(cpl_bundle),intent(inout) :: bun   ! bundle to read
```

9.1.6 cpl_iobin_appendReal – Append real array data to file

Append data in real array `data` to the alredy-open file with unit number `fid`. Include in file the name of the data `vName` and the date `date`. `rcode` is 0 if successful and 1 if file is not open.

REMARKS:

```
This routine is run on root PE only.
```

REVISION HISTORY:

```
2003-Mar-07 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_appendReal(fid,date,vName,data,rcode)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN)   ,intent(in)  :: fid     ! fortan unit number of open data file
type(shr_date),intent(in)  :: date    ! desired model date
character(*)  ,intent(in)  :: vName   ! name of var to read
real(R8)      ,intent(in)  :: data(:) ! the data
integer(IN)   ,intent(out) :: rcode   ! return code
```

9.1.7 cpl_iobin_readReal – read real array data from file

Read data with name `vName` into real array `data` from the alredy-open file with unit number `fid`. Argument `date` is currently not used. `rcode` is 0 if successful and 1 if file is not open or variable is not found.

REMARKS:

```
This routine is run on root PE only.
```

REVISION HISTORY:

```
2002-Nov-08 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_readReal(fid,date,vName,data,rcode)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN)   ,intent(in)  :: fid     ! fortan unit number of open data file
type(shr_date),intent(in)  :: date    ! desired data date
character(*)  ,intent(in)  :: vName   ! name of var to read
real(R8)      ,intent(out) :: data(:) ! array to put data into
integer(IN)   ,intent(out) :: rcode   ! return code
```

9.1.8 cpl_iobin_readDate - read data date from a file

Search the already open file with unit number `fid` and find the date string in the header. If found, return date information in output arguments `cDate` and `sec`. `rcode` is 0 if successful and 1 if file is not open. This routine will abort if the date string is not found. Date is read on node 0 and broadcast to all processors calling this routine.

REMARKS:

```
Assumes the date for all data is the same, hence returns 1st date found.
```

REVISION HISTORY:

```
2002-Dec-13 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iobin_readDate(fid,cDate,sec,rcode)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN),intent(in)  :: fid     ! fortan unit number of open data file
integer(IN),intent(out) :: cDate   ! coded date of data in file
integer(IN),intent(out) :: sec     ! seconds corresponding to cDate
integer(IN),intent(out) :: rcode   ! return code
```

10 NetCDF IO

10.1 Module cpl_iocdf_mod – create and write-to netcdf data file. (Source File: cpl_iocdf_mod.F90)

This module creates and writes data in a `bundle` to a file in machine-independent netCDF format. This module is intended to support the history writing sub-system of cpl6.

REMARKS:

```
Typically this module would be used to support the history file subsystem
of cpl6, but in fact it could be used to create netCDF data files for any
purpose.  The netCDF data file format might provide an alternative to
binary data files.
```

REVISION HISTORY:

```
2002-Oct-22 - B. Kauffman - major upgrade, refactoring
2001-Dec-20 - B. Kauffman - first prototype
```

INTERFACE:

```
module cpl_iocdf_mod
```

USES:

```
    use cpl_mct_mod       ! mct interface
    use cpl_comm_mod      ! mpi/mph communicator info
    use cpl_fields_mod    ! coupler/model data field indicies
    use cpl_bundle_mod    ! defines bundle
    use cpl_domain_mod    ! defines domain
    use cpl_kind_mod      ! defines F90 kinds
    use cpl_const_mod     ! defines constants (eg. spval)
    use cpl_control_mod, only: dbug=>cpl_control_infoDBug
    use shr_sys_mod       ! share system routines
    use shr_date_mod      ! defines date data-type

    implicit none

 #include <netcdf.inc>

    private ! except
```

PUBLIC TYPES:

```
   ! none
```

PUBLIC MEMBER FUNCTIONS:

```
    public :: cpl_iocdf_create   ! create a new file (an empty file)
    public :: cpl_iocdf_open     ! open a named file
    public :: cpl_iocdf_close    ! close an open file
    public :: cpl_iocdf_set64bit ! select 32 or 64 bit real data in file
    public :: cpl_iocdf_append   ! add data to an existing file
```

PUBLIC DATA MEMBERS:

```
    ! none
```

10.1.1 cpl_iocdf_create – create a new file.

Create a new netCDF file with name `fName` and no content other than global attributes. If optional argument `desc` is present, it will be placed in the "description" global attribute.

REVISION HISTORY:

```
2002-Oct-22 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iocdf_create(fName,desc)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)          :: fName   ! file name
character(*),intent(in),optional :: desc    ! description string
```

10.1.2 cpl_iocdf_open – open an existing file.

Open an existing file with name `fName` and return the NetCDF id in the output argument `fid`.

REVISION HISTORY:

```
2002-Oct-22 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iocdf_open(fName,fid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)  :: fName   ! file name
integer(IN) ,intent(out) :: fid     ! file ID
```

10.1.3 cpl_iocdf_close – close a file.

Close the netCDF file with netCDF id `fid`.

REVISION HISTORY:

```
2002-Oct-22 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iocdf_close(fid)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN),intent(in) :: fid     ! file ID
```

10.1.4 cpl_iocdf_set64bit – flags creation of 64 bit netCDF files.

Flags creation of 64 bit netCDF files, default is 32 bit. If argument `flag` is true, netCDF files with be 64 bit.

REMARKS:

```
64 bit netCDF data was introduced for regression testing of coupled system.
```

REVISION HISTORY:

```
2004-Mar-31 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iocdf_set64bit(flag)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
logical,intent(in) :: flag    ! true <=> 64 bit
```

10.1.5 cpl_iocdf_append – add data to an existing file.

Append all data in *bundle* `bun` to an existing netCDF file with netCDF id `fid`. If `fName` is present, use `cpl_iocdf_open` to set `fid`.
If the input time given by the range `dateS, dateE` is not already in the time dimension of the file, it is appended to the end of the time dimension. This algorithm assumes that any input time value not already in the time dimension is greater than any value in the time dimension (so that time will be monotonically increasing).
If `fName` is not present, `fid` must be a valid netCDF file ID for an open netCDF file. This is presumably the normal mode of operation as it can minimize the opening and closing of a file.
If `fName` is present, the named file is opened, data is appended, and the file is closed. In this case, the input `fid` is not used, although it's value will be overwritten. This mode is handy for "one-off" debug files. In either case, the file being appended to must have previously been created.

REVISION HISTORY:

```
2002-Oct-22 - B. Kauffman, initial version
```

INTERFACE:

```
subroutine cpl_iocdf_append(fid,date,bun,dateS,dateE,fName)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
integer(IN)     ,intent(inout)       :: fid       ! file ID
type(cpl_bundle),intent(in)          :: bun       ! bundle
type(shr_date)  ,intent(in)          :: date      ! model date
type(shr_date)  ,intent(in),optional :: dateS     ! time-bound start date
type(shr_date)  ,intent(in),optional :: dateE     ! time-bound end   date
character(*)    ,intent(in),optional :: fName     ! file name
```

11 Communication setup

11.1 Module cpl_comm_mod – Define MPI communication groups and model ID's (Source File: cpl_comm_mod.F90)

Sets up communicator groups and component ID's.
A component ID (CID) is an integer identifying each component in the coupled system. Valid values are 1 to the total number of models (including the Coupler). Declaring an integer for each model is a requirement of using MCT.
Use MPH to define the communicator groups and component ID's.
This module also declares and defines handy data wrt number of pe's, PID's, CID's relative to world and component communicator groups.

REVISION HISTORY:

```
2001-Aug-20 - B. Kauffman - new naming convention
2001-Mar-20 - T. Craig, B. Kauffman, R. Jacob - first prototype
```

INTERFACE:

```
module cpl_comm_mod
```

USES:

```
use cpl_kind_mod   ! kinds
use shr_sys_mod    ! system calls
use shr_mpi_mod    ! mpi layer

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: cpl_comm_init
```

PUBLIC DATA MEMBERS:

```
integer(IN),public :: cpl_comm_wrld         ! = MPI_COMM_WORLD, global comm grp
integer(IN),public :: cpl_comm_wrld_npe     ! number of pe's in MPI_COMM_WORLD
integer(IN),public :: cpl_comm_wrld_pid     ! this comp pid in MPI_COMM_WORLD

integer(IN),public :: cpl_comm_comp         ! this comp communicator group
integer(IN),public :: cpl_comm_comp_npe     ! number of pe's in comp comm group
integer(IN),public :: cpl_comm_comp_pid     ! this comp's pid in comp comm group

integer(IN),public :: cpl_comm_mph_cid      ! MPH component ID, this component
integer(IN),public :: cpl_comm_mph_cid_atm  ! MPH component ID, atm
integer(IN),public :: cpl_comm_mph_cid_ice  ! MPH component ID, ice
integer(IN),public :: cpl_comm_mph_cid_lnd  ! MPH component ID, lnd
integer(IN),public :: cpl_comm_mph_cid_ocn  ! MPH component ID, ocn
integer(IN),public :: cpl_comm_mph_cid_cpl  ! MPH component ID, cpl

integer(IN),public :: cpl_comm_wrld_pe0     ! comm world pe0, this component
```

```
integer(IN),public :: cpl_comm_wrld_pe0_atm ! comm world pe0, atm
integer(IN),public :: cpl_comm_wrld_pe0_ice ! comm world pe0, ice
integer(IN),public :: cpl_comm_wrld_pe0_lnd ! comm world pe0, lnd
integer(IN),public :: cpl_comm_wrld_pe0_ocn ! comm world pe0, ocn
integer(IN),public :: cpl_comm_wrld_pe0_cpl ! comm world pe0, cpl
```

11.1.1 cpl_comm_init – initialize the coupling/mpi environment.

This routine calls *MPI_ init* for the model with name `name` and returns an *MPI_ Communicator* `comm` for use in the calling model. This also sets component ids, and processor ranks relative to world and component communicator groups.

REMARKS:

```
Use cpl_interface_init which calls this routine.
```

REVISION HISTORY:

```
2001-Mar-20 - T. Craig, B. Kauffman, R. Jacob -- first prototype
2001-Dec-10 - R. Jacob -- switch arguments in cpl_mct_world_init to
              to match new version
```

INTERFACE:

```
subroutine cpl_comm_init(name,comm)
```

USES:

```
use cpl_fields_mod      ! contains valid component name strings
use mph_module,only : mph_components
use mph_module,only : mph_global_proc_id
use mph_module,only : mph_local_proc_id
use mph_module,only : mph_total_components
use mph_module,only : mph_comp_id
use mph_module,only : mph_local_totprocs
use mph_module,only : mph_global_totprocs
use mph_module,only : mph_global_id
use mph_module,only : mph_comp_name
use mph_module,only : mph_global_world
use m_MCTWorld    ,only : cpl_mct_world_init   => init

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
character(*),intent(in)  :: name ! name of component name
integer(IN) ,intent(out) :: comm ! communicator group for component
```

12 Constants

12.1 Module cpl_const_mod - defines/provides common constants. (Source File: cpl_const_mod.F90)

Defines/provides common constants.

REVISION HISTORY:

```
2002-jun-10 - B. Kauffman - created module
2002-dec-5  - T. Craig    - names consistent with convention, cpl_const_*
```

INTERFACE:

```
module cpl_const_mod
```

USES:

```
use cpl_kind_mod   ! kinds
use shr_const_mod  ! shared physical constants

implicit none
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
! none
```

PUBLIC DATA MEMBERS:

```
public

!-------------------------------------------------------------------------------
! physical constants
!-------------------------------------------------------------------------------
real(R8),parameter :: cpl_const_pi      = SHR_CONST_PI     ! pi
real(R8),parameter :: cpl_const_rearth  = SHR_CONST_REARTH ! radius of earth ~ m
real(R8),parameter :: cpl_const_rearth2 = SHR_CONST_REARTH*SHR_CONST_REARTH ! rad**2
real(R8),parameter :: cpl_const_g       = SHR_CONST_G      ! gravity
real(R8),parameter :: cpl_const_deg2rad = cpl_const_pi/180.0_R8  ! deg to rads
real(R8),parameter :: cpl_const_rad2deg = 180.0_R8/cpl_const_pi  ! rad to degs

real(R8),parameter :: cpl_const_cpdair  = SHR_CONST_CPDAIR  ! spec heat of dry air
real(R8),parameter :: cpl_const_cpwv    = SHR_CONST_CPWV    ! spec heat of h2o vapor
real(R8),parameter :: cpl_const_cpvir   = cpl_const_cpwv/cpl_const_cpdair - 1.0_R8
real(R8),parameter :: cpl_const_zvir    = SHR_CONST_ZVIR    ! rh2o/rair   - 1.0
real(R8),parameter :: cpl_const_latvap  = SHR_CONST_LATVAP  ! latent heat of evap
real(R8),parameter :: cpl_const_latice  = SHR_CONST_LATICE  ! latent heat of fusion
real(R8),parameter :: cpl_const_stebol  = SHR_CONST_STEBOL  ! Stefan-Boltzmann
real(R8),parameter :: cpl_const_karman  = SHR_CONST_KARMAN  ! Von Karman constant

real(R8),parameter :: cpl_const_ocn_ref_sal = SHR_CONST_OCN_REF_SAL ! ocn ref salt
real(R8),parameter :: cpl_const_ice_ref_sal = SHR_CONST_ICE_REF_SAL ! ice ref salt

real(R8),parameter :: cpl_const_spval       = SHR_CONST_SPVAL       ! special value
```

13 Kind types

13.1 Module cpl_kind_mod – F90 kind declarations (Source File: cpl_kind_mod.F90)

F90 kind declarations.

REVISION HISTORY:

```
2002-Nov-04 - B. Kauffman - created initial version
```

REMARKS:

```
This module does not use the standard cpl6 module variable naming convention
because this would results in excessively long variable declarations.
ie. we want to see real(R8) and not real(cpl_kind_r8)
```

INTERFACE:

```
module cpl_kind_mod
```

USES:

```
use shr_kind_mod  !  shared kind declaration

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
! none
```

PUBLIC DATA MEMBERS:

```
integer,parameter,public :: R16= SHR_KIND_R16 ! 16 byte real
integer,parameter,public :: R8 = SHR_KIND_R8  ! 8 byte real
integer,parameter,public :: R4 = SHR_KIND_R4  ! 4 byte real
integer,parameter,public :: RN = SHR_KIND_RN  ! native/default real
integer,parameter,public :: I8 = SHR_KIND_I8  ! 8 byte integer
integer,parameter,public :: I4 = SHR_KIND_I4  ! 4 byte integer
integer,parameter,public :: IN = SHR_KIND_IN  ! native/default integer

integer,parameter,public :: CL = SHR_KIND_CL  ! generic "long" char string
```

14 MCT Datatypes

14.1 Module cpl_mct_mod – provides a standard API naming convention for MCT code (Source File: cpl_mct_mod.F90)

This module should be used instead of accessing mct modules directly. This module:

- Uses Fortran use renaming of MCT routines and data types so that they all have an mct_ prefix and related data types and routines have related names.
- Provides easy and uniform access to all MCT routines and data types that must be accessed.
- Provides a convienient list of all MCT routines and data types that can be accessed.
- Blocks access to MCT routines that are not used in cpl6.

This module also includes some MCT-only functions to augment the MCT library.

REVISION HISTORY:

```
2001-Aug-14 - B. Kauffman - first prototype
```

INTERFACE:

```
module cpl_mct_mod
```

USES:

```
    use shr_sys_mod          ! share system routines
    use shr_mpi_mod          ! mpi layer
    use cpl_kind_mod         ! kinds
    use cpl_const_mod        ! constant module

    use m_MCTWorld           ,only: cpl_mct_world_init     => init

    use m_AttrVect           ,only: cpl_mct_aVect          => AttrVect
    use m_AttrVect           ,only: cpl_mct_aVect_init     => init
    use m_AttrVect           ,only: cpl_mct_aVect_clean    => clean
    use m_AttrVect           ,only: cpl_mct_aVect_zero     => zero
    use m_AttrVect           ,only: cpl_mct_aVect_lsize    => lsize
    use m_AttrVect           ,only: cpl_mct_aVect_indexIA  => indexIA
    use m_AttrVect           ,only: cpl_mct_aVect_indexRA  => indexRA
    use m_AttrVect           ,only: cpl_mct_aVect_getIList => getIList
    use m_AttrVect           ,only: cpl_mct_aVect_getRList => getRList
    use m_AttrVect           ,only: cpl_mct_aVect_nIAttr   => nIAttr
    use m_AttrVect           ,only: cpl_mct_aVect_nRAttr   => nRAttr
    use m_AttrVect           ,only: cpl_mct_aVect_copy     => Copy
    use m_AttrVectComms      ,only: cpl_mct_aVect_scatter  => scatter
    use m_AttrVectComms      ,only: cpl_mct_aVect_gather   => gather
    use m_AttrVectComms      ,only: cpl_mct_aVect_bcast    => bcast

    use m_Transfer           ,only: cpl_mct_send          => Send
    use m_Transfer           ,only: cpl_mct_recv          => Recv

    use m_GlobalSegMap       ,only: cpl_mct_gsMap         => GlobalSegMap
    use m_GlobalSegMap       ,only: cpl_mct_gsMap_init    => init
    use m_GlobalSegMap       ,only: cpl_mct_gsMap_clean   => clean
    use m_GlobalSegMap       ,only: cpl_mct_gsMap_lsize   => lsize
    use m_GlobalSegMap       ,only: cpl_mct_gsMap_gsize   => gsize

    use m_Rearranger         ,only: cpl_mct_rearr           => Rearranger
```

```
use m_Rearranger         ,only: cpl_mct_rearr_init      => init
use m_Rearranger         ,only: cpl_mct_rearr_clean     => clean
use m_Rearranger         ,only: cpl_mct_rearr_rearrange => rearrange

use m_Router             ,only: cpl_mct_router        => Router
use m_Router             ,only: cpl_mct_router_init   => init
use m_Router             ,only: cpl_mct_router_clean  => clean
use m_SparseMatrixToMaps ,only: cpl_mct_sMat_2XgsMap  => SparseMatrixToXGlobalSegMap
use m_SparseMatrixToMaps ,only: cpl_mct_sMat_2YgsMap  => SparseMatrixToYGlobalSegMap

use m_SparseMatrix       ,only: cpl_mct_sMat              => SparseMatrix
use m_SparseMatrix       ,only: cpl_mct_sMat_Init         => init
use m_SparseMatrix       ,only: cpl_mct_sMat_Vecinit      => vecinit
use m_SparseMatrix       ,only: cpl_mct_sMat_Clean        => clean
use m_SparseMatrix       ,only: cpl_mct_sMat_indexIA      => indexIA
use m_SparseMatrix       ,only: cpl_mct_sMat_indexRA      => indexRA
use m_SparseMatrix       ,only: cpl_mct_sMat_lsize        => lsize
use m_SparseMatrix       ,only: cpl_mct_sMat_nrows        => nRows
use m_SparseMatrix       ,only: cpl_mct_sMat_ncols        => nCols
use m_SparseMatrix       ,only: cpl_mct_sMat_SortPermute  => SortPermute
use m_SparseMatrix       ,only: cpl_mct_sMat_GNumEl       => GlobalNumElements
use m_SparseMatrixComms  ,only: cpl_mct_sMat_ScatterByRow => ScatterByRow
use m_SparseMatrixComms  ,only: cpl_mct_sMat_ScatterByCol => ScatterByColumn
use m_MatAttrVectMul     ,only: cpl_mct_sMat_avMult       => sMatAvMult
use m_GlobalToLocal      ,only: cpl_mct_sMat_g2lMat       => GlobalToLocalMatrix

use m_List               ,only: cpl_mct_list          => list
use m_List               ,only: cpl_mct_list_init     => init
use m_List               ,only: cpl_mct_list_get      => get
use m_List               ,only: cpl_mct_list_nitem    => nitem
use m_List               ,only: cpl_mct_list_clean    => clean
use m_string             ,only: cpl_mct_string        => string
use m_string             ,only: cpl_mct_string_clean  => clean
use m_string             ,only: cpl_mct_string_toChar => toChar
use m_die                ,only: cpl_mct_perr_die      => mp_perr_die

implicit none

This was added specifically for ES compiler bug:
public :: cpl_mct_list, cpl_mct_gsmap, cpl_mct_router
```

14.1.1 cpl_mct_aVect_info - print out aVect info for debugging

Print out information about the input MCT *AttributeVector* `aVect` to stdout. `flag` sets the level of information:

1. print out names of attributes in `aVect`.
2. also print out local max and min of data in `aVect`.
3. also print out global max and min of data in `aVect`.
4. Same as 3 but include name of this routine.

If `flag` is 3 or higher, then optional argument `comm` must be provided. If optional argument `fld` is present, only information for that field will be printed. If optional argument `istr` is present, it will be output before any of the information.

REVISION HISTORY:

```
      2003 Jul 01 - B. Kauffman, T. Craig - first version
```

INTERFACE:

```
 subroutine cpl_mct_aVect_info(flag,aVect,comm,pe,fld,istr)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    integer(IN),intent(in)           :: flag  ! info level flag
    type(cpl_mct_aVect),intent(in)   :: aVect ! Attribute vector
    integer(IN),intent(in),optional  :: comm  ! MPI communicator
    integer(IN),intent(in),optional  :: pe    ! processor number
    character(*),intent(in),optional :: fld   ! fld
    character(*),intent(in),optional :: istr  ! string for print
```

14.1.2 cpl_mct_aVect_getRAttr - get real F90 array data out of an aVect

Get the data associated with attribute `str` in *AttributeVector* `aVect` and return in the real F90 array data `data`. `rcode` will be 0 if succesful, 1 if size of `data` does not match size of `aVect` and 2 if `str` is not found.

REMARKS:

```
     This is like the MCT routine exportRAttr except the output argument
     is not a pointer.
```

REVISION HISTORY:

```
       2002 Apr xx - B. Kauffman - first version
```

INTERFACE:

```
 subroutine cpl_mct_aVect_getRAttr(aVect,str,data,rcode)
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_mct_aVect),intent(in)  :: aVect    ! an Attribute vector
    character(*)       ,intent(in)  :: str      ! field name string
    real(R8)           ,intent(out) :: data(:)  ! an F90 array
    integer(IN)        ,intent(out) :: rcode    ! return code
```

14.1.3 cpl_mct_aVect_putRAttr - put real F90 array data into an aVect

Put the data in array `data` into the *AttributeVector* `aVect` under the attribute `str`. `rcode` will be 0 if succesful, 1 if size of `data` does not match size of `aVect` and 2 if `str` is not found.

REMARKS:

```
     This is like the MCT routine importRAttr except the output argument
     is not a pointer.
```

REVISION HISTORY:

```
      2002 Apr xx - B. Kauffman - first version
```

INTERFACE:

```
 subroutine cpl_mct_aVect_putRAttr(aVect,str,data,rcode)
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_mct_aVect),intent(out) :: aVect ! Attribute vector
    character(*)       ,intent(in)  :: str
    real(R8)           ,intent(in)  :: data(:)
    integer(IN)        ,intent(out) :: rcode
```

14.1.4 cpl_mct_aVect_accum - accumulate attributes from one aVect to another

This routine accumulates from input argment `aVin` into the output *Attr Vect* argument `aVout` the real and integer attributes specified in input `CHARACTER` argument `iList` and `rList`. The attributes can be listed in any order. If neither `iList` nor `rList` are provided, all attributes shared between `aVin` and `aVout` will be copied.
If any attributes in `aVout` have different names but represent the the same quantity and should still be copied, you must provide a translation argument `TrList` and/or `TiList`. The translation arguments should be identical to the `rList` or `iList` but with the correct `aVout` name subsititued at the appropriate place.
N.B.: This routine will fail if the `aVout` is not initialized or if any of the specified attributes are not present in either `aVout` or `aVin`.

REVISION HISTORY:

```
      2002 Sep 15 - ? - initial version.
```

INTERFACE:

```
 subroutine cpl_mct_aVect_accum(aVin, rList, TrList, iList, TiList, aVout)
```

USES:

```
    use m_die ,          only : die
    use m_stdio ,        only : stderr
    use m_String ,       only : String_toChar => toChar
    use m_String ,       only : String
    use m_String ,       only : String_init
    use m_String ,       only : String_clean => clean
    use m_List ,         only : List
    use m_List,          only : List_get => get
    use m_List,          only : List_nullify => nullify
    use m_List,          only : List_clean => clean
    use m_List,          only : init,nitem
    use m_AttrVect,      only : AttrVect
    use m_AttrVect,      only : lsize
    use m_AttrVect,      only : SharedAttrIndexList

    implicit none

  INPUT/OUTPUT PARAMETERS

    type(AttrVect)        ,intent(in)    :: aVin
    character(*), optional,intent(in)    :: iList
    character(*), optional,intent(in)    :: rList
```

```
character(*), optional,intent(in)    :: TiList
character(*), optional,intent(in)    :: TrList
type(AttrVect)        ,intent(inout) :: aVout
```

Part II
Modules used in cpl6 `main`

15 Data Declerations

15.1 Module data_mod – data declaration and initialazion for coupler main. (Source File: data_mod.F90)

Does data declarations and initializations that might otherwise have been located in the coupler main program. See `cpl_bundle_mod`, `cpl_domain_mod`, and `cpl_map_mod` for definitions of the types declared below.

REVISION HISTORY:

```
2002-May-xx - B. Kauffman - added bundleInit & mapInit routines
2002-Apr-28 - B. Kauffman - full set of declarations for CCSM cpl6.0
2001-Jun-08 - T. Craig - first prototype
```

INTERFACE:

```
MODULE data_mod
```

USES:

```
use cpl_mct_mod       ! access to  mct data types
use cpl_domain_mod    ! defines domain data types
use cpl_bundle_mod    ! defines bundle data types
use cpl_map_mod       ! defines map    data types
use cpl_fields_mod    ! indicies into bundles & ibuf
use cpl_control_mod   ! control variables (eg. mapping file names)
use cpl_kind_mod      ! kinds
use shr_sys_mod       ! system call wrappers
use cpl_contract_mod  ! contract

implicit none
```

PUBLIC TYPES:

```
! no public types
```

PUBLIC MEMBER FUNCTIONS:

```
public :: data_bundleInit  ! initialize the bundles declared in this module
public :: data_mapInit     ! initialize the maps    declared in this module
```

PUBLIC DATA MEMBERS:

```
!-----------------------------------------------------------------------------
! datatypes (bundles & routers) for cpl/model communication, both ways
!-----------------------------------------------------------------------------

!--- domains --- includes global grid + local grid and decomp info ---
type(cpl_domain)     :: dom_a   ! atm  domain
type(cpl_domain)     :: dom_i   ! ice  domain
type(cpl_domain)     :: dom_l   ! lnd  domain
type(cpl_domain)     :: dom_r   ! roff domain
type(cpl_domain)     :: dom_o   ! ocn  domain
```

```
!--- bundles to/from component models ---

type(cpl_contract) :: con_Xa2c  ! everything recv'd from atm
type(cpl_contract) :: con_Xl2c  ! everything recv'd from lnd
type(cpl_contract) :: con_Xr2c  ! everything recv'd from runoff
type(cpl_contract) :: con_Xo2c  ! everything recv'd from ocn
type(cpl_contract) :: con_Xi2c  ! everything recv'd from ice

type(cpl_contract) :: con_Xc2a  ! everything sent   to   atm
type(cpl_contract) :: con_Xc2l  ! everything sent   to   lnd
type(cpl_contract) :: con_Dc2l  ! special grid info to   lnd
type(cpl_contract) :: con_Xc2o  ! everything sent   to   ocn
type(cpl_contract) :: con_Xc2i  ! everything sent   to   ice

type(cpl_bundle) :: bun_Xc2oSNAP_o  ! everything sent to ocn, snapshot
type(cpl_bundle) :: bun_Xc2oPSUM_o  ! everything sent to ocn, partial sum

type(cpl_bundle) :: bun_aoflux_o  ! ao fluxes ocn grid
type(cpl_bundle) :: bun_aoflux_a  ! ao fluxes atm grid
type(cpl_bundle) :: bun_oalbedo_o ! ocean albedos on ocn grid
type(cpl_bundle) :: bun_oalbedo_a ! ocean albedos on atm grid
type(cpl_bundle) :: bun_precip_o  ! total snow and rain on ocn grid
type(cpl_bundle) :: bun_precip_a  ! total snow and rain on atm grid

type(cpl_bundle) :: bun_Sa2c_a  ! a2c states
type(cpl_bundle) :: bun_Fa2c_a  ! a2c fluxes
type(cpl_bundle) :: bun_Sa2c_o  ! a2c states mapped to o
type(cpl_bundle) :: bun_Fa2c_o  ! a2c fluxes mapped to o
type(cpl_bundle) :: bun_Sl2c_l  ! l2c states
type(cpl_bundle) :: bun_Fl2c_l  ! l2c fluxes
type(cpl_bundle) :: bun_Sl2c_o  ! l2c states mapped to o
type(cpl_bundle) :: bun_Fl2c_o  ! l2c fluxes mapped to o
type(cpl_bundle) :: bun_Xr2c_o  ! r2c fields mapped to o
type(cpl_bundle) :: bun_So2c_o  ! o2c states
type(cpl_bundle) :: bun_Fo2c_o  ! o2c fluxes
type(cpl_bundle) :: bun_So2c_a  ! o2c states mapped to a
type(cpl_bundle) :: bun_Fo2c_a  ! o2c fluxes mapped to a
type(cpl_bundle) :: bun_Si2c_i  ! i2c states
type(cpl_bundle) :: bun_Fi2c_i  ! i2c fluxes
type(cpl_bundle) :: bun_Si2c_a  ! i2c states mapped to a
type(cpl_bundle) :: bun_Fi2c_a  ! i2c fluxes mapped to a

!--- fundamental maps ---
type(cpl_map),target :: map_Sa2o  ! maps states a->o grids
type(cpl_map),target :: map_Fa2o  ! maps fluxes a->o grids
type(cpl_map),target :: map_So2a  ! maps states o->a grids
type(cpl_map),target :: map_Fo2a  ! maps fluxes o->a grids
type(cpl_map),target :: map_Xr2o  ! maps fluxes r->o grids
type(cpl_map),target :: map_ID    ! identity map

!--- redundant maps ---
type(cpl_map),pointer :: map_Fa2i  ! maps fluxes a->i grids
type(cpl_map),pointer :: map_Fa2l  ! maps fluxes a->l grids
type(cpl_map),pointer :: map_Sa2i  ! maps states a->i grids
type(cpl_map),pointer :: map_Sa2l  ! maps states a->l grids

type(cpl_map),pointer :: map_Fi2a  ! maps fluxes i->a grids
type(cpl_map),pointer :: map_Fi2l  ! maps fluxes i->l grids
```

```
type(cpl_map),pointer :: map_Fi2o  ! maps fluxes i->o grids
type(cpl_map),pointer :: map_Si2a  ! maps states i->a grids
type(cpl_map),pointer :: map_Si2l  ! maps states i->l grids
type(cpl_map),pointer :: map_Si2o  ! maps states i->o grids

type(cpl_map),pointer :: map_Fl2a  ! maps fluxes l->a grids
type(cpl_map),pointer :: map_Fl2i  ! maps fluxes l->i grids
type(cpl_map),pointer :: map_Fl2o  ! maps fluxes l->o grids
type(cpl_map),pointer :: map_Sl2a  ! maps states l->a grids
type(cpl_map),pointer :: map_Sl2i  ! maps states l->i grids
type(cpl_map),pointer :: map_Sl2o  ! maps states l->o grids

type(cpl_map),pointer :: map_Fo2i  ! maps fluxes o->i grids
type(cpl_map),pointer :: map_Fo2l  ! maps fluxes o->l grids
type(cpl_map),pointer :: map_So2i  ! maps states o->i grids
type(cpl_map),pointer :: map_So2l  ! maps states o->l grids

save
```

15.1.1 data_bundleInit - initialize all bundles

Initialize all the bundle's declared as module variables above.

REVISION HISTORY:

```
2002-Mar-06 - B. Kauffman - first version
```

INTERFACE:

```
subroutine data_bundleInit()
```

USES:

INPUT/OUTPUT PARAMETERS:

```
! input/output are all variables declared in this module
```

15.1.2 data_mapInit - initialize all mapping data

Initialize all the mapping data by calling `cpl_map_init` for the 5 maps currently needed in CCSM3. Map filenames are obtained from `cpl_control_map*` module variables which are initialized from the Coupler namelist.

REVISION HISTORY:

```
2002-May-21 - B. Kauffman - first version
```

INTERFACE:

```
subroutine data_mapInit()
```

USES:

INPUT/OUTPUT PARAMETERS:

```
! output are the cpl_map data types declared in this module
```

16 Flux Calculations

16.1 Module flux_mod – Coupler's flux calculations. (Source File: flux_mod.F90)

The coupler is required to do certain flux calculations – those calculations are located in this module. Details about the calculations performed can be found in Part III of the Cpl6 Combined User's Guide, Source Code Reference and Scientific Description.

REVISION HISTORY:

```
2002-Jun-10 - B. Kauffman - first version
```

INTERFACE:

```
module flux_mod
```

USES:

```
use shr_sys_mod     ! shared system routines
use shr_date_mod    ! shared date module
use shr_mpi_mod     ! shared mpi layer
use cpl_kind_mod    ! kinds
use cpl_const_mod   ! constants module
use cpl_mct_mod     ! mct library
use cpl_comm_mod    ! communicator module
use cpl_fields_mod  ! list of fields found in bundles
use cpl_domain_mod  ! domain data types
use cpl_bundle_mod  ! bundle data types
use cpl_control_mod, dbug=>cpl_control_infoDBug

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: flux_atmOcn  ! computes atm/ocn fluxes
public :: flux_albo    ! computes ocn albedos
public :: flux_albi    ! modifies ice reference albedo
public :: flux_solar   ! computes ocn net solar
public :: flux_epbal   ! forces evap/precip/runoff balance
```

PUBLIC DATA MEMBERS:

```
! none
```

16.1.1 flux_atmOcn - wrapper to atm/ocn flux calculation

Using data from the ocean model in `bun_ocn` and data from the atmosphere model (already mapped to the ocean grid) in `bun_atm`, calculate the ocean-atmosphere fluxes and return them in `bun_flux`.
If optional argument `fabricate` is present, this routine will use pre-set values instead of the input data. This is useful when using dead models.

REMARKS:

```
All data must be on the ocean domain.
```

REVISION HISTORY:

```
2002-Jun-10 - B. Kauffman - first version
```

INTERFACE:

```
subroutine flux_atmOcn(bun_ocn,bun_atm,fabricate,bun_flux)
```

USES:

```
use shr_timer_mod
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(in ) :: bun_ocn    ! ocn state fields on ocn domain
type(cpl_bundle),intent(in ) :: bun_atm    ! atm state fields on ocn domain
logical         ,intent(in ) :: fabricate  ! T <=> fabriate/clobber input data
type(cpl_bundle),intent(out) :: bun_flux   ! flux fields on ocn grid
```

16.1.2 flux_albo - ocean albedo calculation

Depending on choices below, alter the abledos in the input `bun_ocn`
If Coupler namelist variable **flux_albav = .true.**
Compute four effective daily avg surface albedos for all combinations of visible/near-infrared and direct/diffuse radiation without accounting for zenith angle (ie. a "daily average" albedo)

If **flux_albav = .false.** then
Compute four surface albedos for all combinations of visible/ near-infrared and direct/diffuse radiation, accounting for instantaneous zenith angle calculated from input argument `date` and the following:

- upon input, albedos are assumed to be a 60 degree reference albedo
- albedos are computed by taking the 60 deg reference albedo and then adjusting this value based on zenith angle
- Albedos are independent of spectral interval and other physical factors such as surface wind speed.

For more details see Briegleb, Bruce P., 1992: "Delta-Eddington Approximation for Solar Radiation in the NCAR Community Climate Model", *Journal of Geophysical Research*, Vol. **97**, D7, pp7603-7612.

REVISION HISTORY:

```
198x        - CCM1, original version
1992-Jun    - J. Rosinski -- standardized
1994-May    - J. Rosinski -- rewritten for land only
1994-Jul    - B. Kauffman -- rewritten for ocean only
2002-Oct-26 - R. Jacob -- Rewritten for cpl6
```

INTERFACE:

```
subroutine flux_albo(date,bun_ocn)
```

USES:

```
use shr_orb_mod ! orbital constants and methods

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout ) :: bun_ocn ! ocn albedo fields
type(shr_date)  ,intent(in )    :: date    ! current date
```

16.1.3 flux_albi - ice albedo modification

Depending on choices below, alter the albedos in the input `bun_ice`
If Coupler namelist variable **flux_albav = .false.**,
impose a zenith angle dependance on the ice model "reference albedo". Currently this only involves setting albedos to zero on the dark side of the earth. Use input `date` to determine zenith angle.

If Coupler namelist variable **flux_albav = .true.**,
do not alter ice albedos. Ice albedos are zenith-angle independent

REMARKS:

```
o upon input, albedos are assumed to be a 60 degree reference albedo
```

REVISION HISTORY:

```
199x-       - B. Kauffman -- original cpl5 version
2002-Oct-26 - R. Jacob -- rewritten for cpl6
```

INTERFACE:

```
subroutine flux_albi(date,bun_ice)
```

USES:

```
use shr_orb_mod

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(inout)  :: bun_ice ! contains ice albedo fields
type(shr_date)  ,intent(in )    :: date    ! current date
```

16.1.4 flux_solar - compute atm/ocn absorbed short-wave (net sw)

Use the downward shortwave radiation in `bun_atm` and the albedos in `bun_ocn` and compute the atm/ocn absorbed short-wave (net sw) and return it in `bun_aoflux`.

REVISION HISTORY:

```
2000-Jan-03 - B. Kauffman -- original cpl5 version
2002-Oct-26 - R. Jacob -- rewritten for cpl6
```

INTERFACE:

```
subroutine flux_solar(bun_atm,bun_ocn,bun_aoflux)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(cpl_bundle),intent(in  ) :: bun_atm    ! contains atm sw down fields
type(cpl_bundle),intent(in  ) :: bun_ocn    ! contains ocn albedo fields
type(cpl_bundle),intent(out ) :: bun_aoflux ! contains a/o net-sw fields
```

16.1.5 flux_epbal - Calculate precip/runoff adjustment factor

If Coupler namelist varible `flx_epbal` is not "off", adjust precip (an atm output flux) and runoff (a lnd output) sent to ice & ocn by a scalar factor f, so that

$$P' + R' + E = f(P + R) + E = 0$$

This will insure a net zero fresh-water flux into ocn+ice. This could be used to compensate for fresh-water flux imbalances, e.g. due to the lack of river runoff from the lnd model.
if `flx_epbal` = "ocn", factor f must be sent by the ocean model in the *infobuffer*.
if `flx_epbal` = "inst", factor f will be calculated by this routine.
`bun_aoflux` contains the atm/ocean evaporation, `bun_i2c` the ice/atm evaporation, `bun_prec` contains the total snow and rain, `bun_r2c` contains the runoff and `bun_frac` contains the ocean domain surface fractions. If adjustment is performed, on return the runoff in `bun_r2c` and rain and snow in `bun_prec` will be altered to balance the evaporation.

REVISION HISTORY:

```
199x        - B. Kauffman -- Original cpl5 version
2003-Feb-17 - R. Jacob -- rewritten for cpl6
```

INTERFACE:

```
subroutine flux_epbal(date,bun_aoflux,bun_i2c,bun_prec,bun_r2c,bun_frac)
```

USES:

```
implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(shr_date)  ,intent(in   ) :: date       ! current date
type(cpl_bundle),intent(in   ) :: bun_i2c    ! ice to cpl bundle: ice evap
type(cpl_bundle),intent(in   ) :: bun_aoflux ! a/o flux bundle  : ocn evap
type(cpl_bundle),intent(inout) :: bun_prec   ! a/x precip bundle: i+o prec
type(cpl_bundle),intent(inout) :: bun_r2c    ! runoff bundle    : ocn roff
type(cpl_bundle),intent(in   ) :: bun_frac   ! fractions on ocn domain
```

17 Time Coordination

17.1 timeCheck – verify/enforce component time coordination. (Source File: timeCheck.F

Verify/enforce component time coordination. The Coupler's date in `date_c` is checked against the date in the *infobuf* of the most recently received *contract* for each model.
If time is uncoordinated, print out information. If time is coordinated and `print` is true, print out Coupler date.
If `enforce` is true, abort if time's are not coordinated.

REMARKS:

```
date/time for component models is "global" data accessed from module data_mod.
```

REVISION HISTORY:

```
2002-Nov-20 - B. Kauffman - initial version.
```

INTERFACE:

```
subroutine timeCheck(date_c,print,enforce)
```

USES:

```
use cpl_kind_mod        ! access to F90 kind declarations
use shr_cal_mod         ! access to calendar routines
use shr_date_mod        ! access to date     routines
use data_mod            ! lengthly data declarations/inits for main program
use shr_sys_mod         ! wrappers to system calls

implicit none
```

INPUT/OUTPUT PARAMETERS:

```
type(shr_date),intent(in)  :: date_c  ! official coupler date
logical       ,intent(in)  :: print   ! true => print out the cpl date
logical       ,intent(in)  :: enforce ! true => abort if time uncoordinated
```

18 Diagnostics

18.1 Module diag_mod – computes spatial & time averages of fluxed quatities (Source File: diag_mod.F90)

The coupler is required to do certain diagnostics, those calculations are located in this module.

REMARKS:

```
Sign convention:
   positive value <=> the model is gaining water, heat, momentum, etc.
Unit convention:
   heat flux     ~ W/m^2
   momentum flux ~ N/m^2
   water flux    ~ (kg/s)/m^2
   salt  flux    ~ (kg/s)/m^2
```

REVISION HISTORY:

```
199x-mmm-dd - B. Kauffman - original cpl5 version
2002-nov-21 - R. Jacob - initial port to cpl6. Does atm and lnd
2002-nov-27 - R. Jacob - add ocean
2002-dec-03 - R. Jacob - add solar diagnostics
2002-dec-15 - R. Jacob - time average diagnostics
2003-Feb-10 - R. Jacob - calculate sums locally
```

INTERFACE:

```
module diag_mod
```

USES:

```
use shr_date_mod    ! shared date module
use shr_sys_mod     ! shared system routines
use shr_timer_mod   ! shared timers
use shr_mpi_mod     ! shared mpi layer
use cpl_kind_mod    ! kinds
use cpl_const_mod   ! physical constants
use cpl_mct_mod     ! mct library
use cpl_comm_mod    ! communicator module
use cpl_fields_mod  ! index to fields in bundles
use cpl_domain_mod  ! domain data types
use cpl_bundle_mod  ! bundle data types
use cpl_control_mod ! cpl control flags & methods

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: diag_doDiag   ! coordinates all diagnostic subroutines
public :: diag_solar    ! verifies net-solar coordination
```

PUBLIC DATA MEMBERS:

```
!--- note: this partial-sum data needs to be saved in a restart file ---
real(R8),save,public        :: diag_eday0        ! partial sum: start date
real(R8),save,public        :: diag_eday1        ! partial sum: end   date
real(R8),save,public        :: diag_ns           ! partial sum: number of samples
real(R8),save,public,target :: diag_datas(8,6,3) ! partial sum: the p-sum data
```

18.1.1 diag_doDiag - coordinates calculation of diagnostic data

Calculate global diagnostics.

REMARKS:

```
if (cpl_control_diagNow  ) then print instantaneous diagnostics
if (cpl_control_avDiagNow) then print time-avg diagnostics
This is hard-coded to print/reset the t-avg data at the end of every year.
```

REVISION HISTORY:

```
199x-mmm-dd - B. Kauffman original cpl5 version, called diagnos in diag_mod
```

INTERFACE:

```
subroutine diag_doDiag(date,bun_a2c,bun_c2a ,bun_l2c ,bun_c2l ,bun_r2c  , &
          &                bun_i2c,bun_c2i ,bun_o2c ,bun_c2o ,bun_a2c_o, &
          &                bun_alb,bun_lfrac,bun_ifrac,bun_ofrac)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(shr_date)  ,intent(in) :: date       ! current model date
type(cpl_bundle),intent(in) :: bun_a2c    ! atm->cpl bundle
type(cpl_bundle),intent(in) :: bun_c2a    ! cpl->atm bundle
type(cpl_bundle),intent(in) :: bun_l2c    ! lnd->cpl bundle
type(cpl_bundle),intent(in) :: bun_c2l    ! cpl->lnd bundle
type(cpl_bundle),intent(in) :: bun_r2c    ! rof->cpl bundle
type(cpl_bundle),intent(in) :: bun_i2c    ! ice->cpl bundle
type(cpl_bundle),intent(in) :: bun_c2i    ! cpl->ice bundle
type(cpl_bundle),intent(in) :: bun_o2c    ! ocn->cpl bundle
type(cpl_bundle),intent(in) :: bun_c2o    ! cpl->ocn bundle
type(cpl_bundle),intent(in) :: bun_a2c_o  ! atm->cpl bundle
type(cpl_bundle),intent(in) :: bun_alb    ! albedo   bundle

type(cpl_bundle),intent(in) :: bun_lfrac  ! surface fractions on lnd domain
type(cpl_bundle),intent(in) :: bun_ifrac  ! surface fractions on ice domain
type(cpl_bundle),intent(in) :: bun_ofrac  ! surface fractions on ocn domain
```

18.1.2 diag_atm - compute atmosphere diagnostics

Compute atmosphere diagnostics (instantaneous global averages)

REMARKS:

```
Area averages are relative to the entire unit sphere, area = 4*pi rad^2
```

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_atm(bun_a2c,bun_c2a)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(cpl_bundle),intent(in ) :: bun_a2c ! atm->cpl bundle
type(cpl_bundle),intent(in ) :: bun_c2a ! cpl->atm bundle
```

18.1.3 diag_lnd - compute land diagnostics

Compute land diagnostics (instantaneous global averages)

REMARKS:

```
Area averages are relative to the entire unit sphere, area = 4*pi rad^2
```

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_lnd(bun_l2c,bun_c2l,bun_r2c,bun_lfrac)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(cpl_bundle),intent(in) :: bun_l2c   ! lnd->cpl bundle
type(cpl_bundle),intent(in) :: bun_c2l   ! cpl->lnd bundle
type(cpl_bundle),intent(in) :: bun_r2c   ! rof->cpl bundle
type(cpl_bundle),intent(in) :: bun_lfrac ! surface fractions on lnd domain
```

18.1.4 diag_ice - compute atmosphere diagnostics

Compute ice model diagnostics (instantaneous global averages)

REMARKS:

```
Area averages are relative to the entire unit sphere, area = 4*pi rad^2
This routine assumes ice and ocean are on the same grid.
```

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_ice(bun_i2c,bun_c2i,bun_o2c,bun_ifrac)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(cpl_bundle),intent(in ) :: bun_i2c   ! ice->cpl bundle
type(cpl_bundle),intent(in ) :: bun_c2i   ! cpl->ice bundle
type(cpl_bundle),intent(in ) :: bun_o2c   ! ocn->cpl bundle
type(cpl_bundle),intent(in ) :: bun_ifrac ! surface fractions on ice domain
```

18.1.5 diag_ocn - compute ocean diagnostics

Compute ocean diagnostics (instantaneous global averages)

REMARKS:

```
Area averages are relative to the entire unit sphere, area = 4*pi rad^2
```

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_ocn(bun_o2c,bun_c2o,bun_a2c,bun_i2c,bun_alb,bun_ofrac)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(cpl_bundle),intent(in ) :: bun_o2c    ! ocn->cpl bundle
type(cpl_bundle),intent(in ) :: bun_c2o    ! cpl->ocn bundle
type(cpl_bundle),intent(in ) :: bun_a2c    ! atm->cpl bundle
type(cpl_bundle),intent(in ) :: bun_i2c    ! ice->cpl bundle
type(cpl_bundle),intent(in ) :: bun_alb    ! albedo   bundle
type(cpl_bundle),intent(in ) :: bun_ofrac  ! surface fractions on ocn domain
```

18.1.6 diag_print - print out diagnostics

Print the diagnostics and their sum in each category

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_print(date)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(shr_date),intent(in)  :: date ! current model date
```

18.1.7 diag_printAvg - print out diagnostics for time-avg data

Print the diagnostics and their sum in each category

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_printAvg(date)
```

USES:

```
implicit none
```

INPUT PARAMETERS:

```
type(shr_date),intent(in)  :: date ! current model date
```

18.1.8 diag_solar - compares expected vs. actual short-wave radiation

Compare expected vs. actual short-wave net (absorbed solar)

REVISION HISTORY:

INTERFACE:

```
SUBROUTINE diag_solar(bun_a2c,bun_l2c,bun_i2c,bun_lfrac,bun_ifrac)
```

INPUT PARAMETERS:

```
implicit none

type(cpl_bundle),intent(in) :: bun_a2c   !  atm->cpl bundle
type(cpl_bundle),intent(in) :: bun_l2c   !  lnd->cpl bundleand data
type(cpl_bundle),intent(in) :: bun_i2c   !  ice->cpl bundlece data
type(cpl_bundle),intent(in) :: bun_lfrac !  surface fractions on lnd domain
type(cpl_bundle),intent(in) :: bun_ifrac !  surface fractions on ice domain
```

19 Merging

19.1 Module merge_mod – field merging module. (Source File: merge_mod.F90)

Merges fields to be sent to a component. "Merging" means combining one or more fields to create a new field. Typically this is two or more fields of the same type, e.g. combining atm/ice, atm/lnd, and atm/ocn sensible heat flux fields, weighted by surface fraction, to create an atm/surface heat flux. But it could also involve somewhat differing fields, e.g. precipitation plus snow melt. Merging is normally not an automatic process, some hand-tuning is generally necessary to achieve the results necessary for valid science.

REVISION HISTORY:

```
2002-Sep-27 - B. Kauffman - created initial version
```

INTERFACE:

```
module merge_mod
```

USES:

```
use cpl_kind_mod        ! kinds
use cpl_control_mod     ! control flags
use cpl_bundle_mod      ! bundle data type and methods
use frac_mod            ! surface fractions
use data_mod            ! lengthly data declarations/inits for main program
use shr_sys_mod         ! wrappers to system calls
use shr_timer_mod       ! timing utilities

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: merge_atm        ! merges atm input data
public :: merge_ocn        ! merges ocn input data
```

PUBLIC DATA MEMBERS:

```
! none
```

19.1.1 merge_atm – merge bundles to form atm input bundle

Merge *bundles* from multiple sources to form the atm input *bundle* in `con_Xc2a%bundle`.
First a `cpl_bundle_gather` is done to copy fields with the same name into `con_Xc2a%bundle`. Then the field-specific merges are performed.

REMARKS:

```
The SST, field "So_t", is copied from input argument fix_So2c_a
to avoid sub-freezing values introduced by mapping.
```

REVISION HISTORY:

```
       2002-Jun-09 - B. Kauffman - initial version.
```

INTERFACE:

```
 subroutine merge_atm(fix_So2c_a)

    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_bundle),intent(in) :: fix_So2c_a ! KLUDGE: use alt bun_So2a_a, temp fix
```

19.1.2 merge_ocn – merge bundles to form ocn input bundle

Merge *bundles* from multiple sources to form the "snapshot" ocn input *bundle* `bun_Xc2oSNAP_o`. This "snapshot" will be added to the running sum which will be averaged when time to communicate with the ocean. First a `cpl_bundle_gather` is done then the field-specific merges are performed.

REMARKS:

REVISION HISTORY:

```
       2002-Jun-06 - B. Kauffman - initial version.
```

INTERFACE:

```
 subroutine merge_ocn()

    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    ! none. uses module data in data_mod
```

20 Area Normalizing

20.1 Module areafact_mod – Handle normalization area fractions. (Source File: areafact_mod.F90)

Defines, declares, initializes, and updates *bundles* for area normalizing.
These are used to correct flux fields received and sent to components based on differences between grid cell areas in the models and the areas in the mapping weights files.

REVISION HISTORY:

```
2003-Jan-02 - T. Craig, 1st version.
2003-Jan-06 - T. Craig, moved work to areafact_set, removed use of cpl_map
```

INTERFACE:

```
module areafact_mod
```

USES:

```
use shr_sys_mod         ! share system routines
use cpl_kind_mod        ! kinds
use cpl_mct_mod         ! mct routines
use cpl_comm_mod        ! comms
use cpl_domain_mod      ! defines domain
use cpl_bundle_mod      ! defines bundle
use cpl_control_mod     ! control paramters

implicit none

private ! except
```

PUBLIC TYPES:

```
! no public types
```

PUBLIC MEMBER FUNCTIONS:

```
public :: areafact_init
```

PUBLIC DATA MEMBERS:

```
type(cpl_bundle),public :: bun_areafact_a   ! area corrections
type(cpl_bundle),public :: bun_areafact_l   ! area corrections
type(cpl_bundle),public :: bun_areafact_o   ! area corrections
type(cpl_bundle),public :: bun_areafact_i   ! area corrections
type(cpl_bundle),public :: bun_areafact_r   ! area corrections

character(*),parameter :: bun_areafact_fields =  'cpl2comp:comp2cpl'
```

20.1.1 areafact_init - Initialize all area factor bundles

Initialize the `bun_areafact_*` *bundles* declared in `data_mod.F90`. All fractions are derived from the (time-invariant) component model areas and the time-invariant mapping weights areas contained in the input *domains*.

REVISION HISTORY:

```
      2003-Jan-02 - T. Craig, 1st version.
```

INTERFACE:

```
 subroutine areafact_init(domain_a,domain_i,domain_l,domain_o,domain_r)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
    type(cpl_domain),intent(in) :: domain_a ! domain of atm bundle
    type(cpl_domain),intent(in) :: domain_i ! domain of ice bundle
    type(cpl_domain),intent(in) :: domain_l ! domain of lnd bundle
    type(cpl_domain),intent(in) :: domain_o ! domain of ocn bundle
    type(cpl_domain),intent(in) :: domain_r ! domain of runoff bundle
```

21 Surface Fractions

21.1 Module frac_mod – handles surface fractions. (Source File: frac_mod.F90)

Defines, declares, initializes, and updates surface fractions.
These fractions are used for merging fields onto various domains. This particular implementation of this module makes certain assumptions about which domains exist and the relationships between them. These assumptions are hard-coded into this software implementation.
ASSUMPTIONS:

- atm and lnd grid cells and domain decompostion are identical
- ice and ocn domains are identical (includes decomposition)
- all atm cells are fully active
- all ocn cells are either fully active or fully inactive
- lnd cells can be partially active – the fraction of a lnd cell that is active is the fraction that is not occupied by ocn
- ice cells can be partially active – the fraction that is active is determined by the ice component itself

For each domain (atm,lnd,ice,ocn) there are four fractions: f_a, f_i, f_l, f_o, three that could be used for merging, and one which indicatates the fraction of the cell which is active.

- merging on atm domain: $F_a = f_i F_i + f_l F_l + f_o F_o$ $\quad (f_i + f_l + f_o = 1)$
- merging on ice domain: $F_i = f_a F_a + f_o F_o = F_a + F_o$ $\quad (f_a = f_o = 1, f_l = 0)$
- merging on lnd domain: $F_l = f_a F_a = F_a$ $\quad (f_a = 1, f_o = f_i = 0)$
- merging on ocn domain: $F_o = f_a F_a + f_o F_o$ $\quad (f_a + f_i = 1, f_l = 0)$
- on the atm domain: $f_a = 1$ (atm cells are fully active)
- on the ice domain: f_i is time-variant and determined by the ice model
- on the lnd domain: $f_l = 1 - f_o$ and is time-invariant
- on the ocn domain: $f_o = 1$ (ocn cells are fully active)

REVISION HISTORY:

```
2002-Aug-21 - B. Kauffman, 1st version.
```

INTERFACE:

```
module frac_mod
```

USES:

```
use shr_sys_mod         ! shared system routines
use shr_timer_mod       ! shared timer routines
use shr_mpi_mod         ! shared mpi layer
use cpl_kind_mod        ! kinds
use cpl_comm_mod        ! mpi/mph communicator info
use cpl_mct_mod         ! mct interface
use cpl_const_mod       ! constants
use cpl_domain_mod      ! defines domain
use cpl_bundle_mod      ! defines bundle
use cpl_map_mod         ! access to map data types and methods
use cpl_control_mod, only: dbug=>cpl_control_infoDBug
```

```
implicit none

private ! except
```

PUBLIC TYPES:

```
! no public types
```

PUBLIC MEMBER FUNCTIONS:

```
public :: frac_init  ! one-time initialization of fraction values
public :: frac_set   ! time-variant update of fraction values
```

PUBLIC DATA MEMBERS:

```
!--- note: these could be declared in data_mod.F90           ---
!--- or in a cpl/frac_mod.F90 & passed down from main program ---

type(cpl_bundle),public :: bun_frac_a   ! surface fractions on atm domain
type(cpl_bundle),public :: bun_frac_i   ! surface fractions on ice domain
type(cpl_bundle),public :: bun_frac_l   ! surface fractions on lnd domain
type(cpl_bundle),public :: bun_frac_o   ! surface fractions on ocn domain

character(*),parameter :: frac_fields = 'afrac:ifrac:lfrac:ofrac'
```

21.1.1 frac_init - Initialize all the surface fraction bundles

Initialize all the fraction *bundles* `bun_frac_*` using the input *domains* `domain_`. All fractions are derived from the (time-invariant) ice/ocn domain masks plus the (time-variant) ice fraction.
This initialization routine sets the time-invariant values.

REVISION HISTORY:

```
2002-aug-21 - B. Kauffman, 1st version.
```

INTERFACE:

```
subroutine frac_init(map_o2a,domain_a,domain_i,domain_l,domain_o)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
type(cpl_map   ),intent(inout) :: map_o2a  ! use to map ice-frac from ocn -> atm
type(cpl_domain),intent(in   ) :: domain_a ! domain of atm fraction bundle
type(cpl_domain),intent(in   ) :: domain_i ! domain of ice fraction bundle
type(cpl_domain),intent(in   ) :: domain_l ! domain of lnd fraction bundle
type(cpl_domain),intent(in   ) :: domain_o ! domain of ocn fraction bundle
```

21.1.2 frac_set - set/update the surface fraction bundles

Set/update the fraction bundles to account for time varying sea-ice distribution using the ice fraction information in the input real array `ifrac_i`.
Update `bun_frac_i`, then `bun_frac_o`, then use `map_o2a` to update `bun_frac_a`. Make use of `bun_frac_l`.
The companion initialization routine `frac_init` must be called first to set the time-invariant values.

REMARKS:

```
The domain_* arguments should be removed because they aren't used.
```

REVISION HISTORY:

```
2002-aug-21 - B. Kauffman, 1st version.
```

INTERFACE:

```
subroutine frac_set(ifrac_i,map_o2a,domain_a,domain_i,domain_l,domain_o)
```

USES:

INPUT/OUTPUT PARAMETERS:

```
real(R8)        ,intent(in   ) :: ifrac_i(:) ! temporary data array on atm domain
type(cpl_map   ),intent(inout) :: map_o2a    ! use to map ifrac from ocn -> atm
type(cpl_domain),intent(in   ) :: domain_a   ! domain of atm fraction bundle
type(cpl_domain),intent(in   ) :: domain_i   ! domain of ice fraction bundle
type(cpl_domain),intent(in   ) :: domain_l   ! domain of lnd fraction bundle
type(cpl_domain),intent(in   ) :: domain_o   ! domain of ocn fraction bundle
```

22 History Writing

22.1 Module history_mod – cpl6 main program history file creation module. (Source File: history_mod.F90)

cpl6 main program history file creation module. Contains routine to coordinate the writing of history files.

REVISION HISTORY:

```
2002-Sep-27 - B. Kauffman - created initial version
```

REMARKS:

```
This is not a generic low-level module, it is a high-level module
hard-coded to particular bundle declarations and user desires wrt history
file content.
```

INTERFACE:

```
module history_mod
```

USES:

```
use cpl_control_mod     ! control flags
use shr_date_mod        ! date/time module
use cpl_iocdf_mod       ! netCDF file creation
use frac_mod            ! surface fractions
use areafact_mod        ! area corrections
use cpl_kind_mod        ! kinds
use data_mod            ! lengthly data declarations/inits for main program
use shr_sys_mod         ! wrappers to system calls
use shr_timer_mod       ! timing utilities


implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: history_write    ! create cpl6 history files
public :: history_avbundleInit  ! initialize cpl6 tavg history bundles
public :: history_avwrite  ! create cpl6 tavg history files
```

PUBLIC DATA MEMBERS:

```
! none
```

22.1.1 history_write – Write history file with preset contents.

Create desired history file from most of the data in data_mod data and include input date `date`. Writes out contents of `bun_areafact_`, `bun_frac_*` and the bundles in the x2c and c2x *contracts*.
If `cpl_control_histNow` is false, this routine does nothing.

REVISION HISTORY:

2002-Sep-27 - B. Kauffman - initial version.

INTERFACE:

```
subroutine history_write(date)

   implicit none
```

INPUT/OUTPUT PARAMETERS:

```
   type(shr_date) :: date ! date associated with bundles
```

22.1.2 history_avbundleInit – Initialize bundles for time-average data.

Initialize the `bun_avX*` bundles used to make time-averaged data.

REVISION HISTORY:

2003-Mar-31 - T. Craig - initial version

INTERFACE:

```
subroutine history_avbundleInit()

   implicit none
```

INPUT/OUTPUT PARAMETERS:

22.1.3 history_avwrite – Accumulate data in history bundles and/or write out.

Add current values of desired data to the `bun_avX*` bundles using input `date`.
If `cpl_control_avhistNow` is true, form time average and write out data. Then zero the `bun_avX*` bundles.
If `cpl_control_avhistType` is none, this routine does nothing.

REVISION HISTORY:

2003-Mar-30 - T. Craig - initial version.

INTERFACE:

```
subroutine history_avWrite(date)

   implicit none
```

INPUT/OUTPUT PARAMETERS:

```
   type(shr_date) :: date ! date associated with bundles
```

23 Restart Writing

23.1 Module restart_mod – support cpl6 main program read/write restart files. (Source File: restart_mod.F90)

Routines for reading and writing the cpl6 restart file.

REVISION HISTORY:

```
2002-Nov-06 - B. Kauffman - created initial version
```

REMARKS:

```
This is not a generic low-level module, it is a high-level module
hard-coded to particular bundle declarations and restart data needs for a
particular version of the coupler.
```

INTERFACE:

```
module restart_mod
```

USES:

```
use cpl_kind_mod        ! access to F90 kind declarations
use cpl_control_mod, dbug=>cpl_control_infoDBug
use cpl_comm_mod        ! mpi communicator groups & related
use diag_mod            ! runtime diagnostic subsystem module
use shr_date_mod        ! date/time module
use cpl_iobin_mod       ! binary data file creation
use frac_mod            ! surface fractions
use data_mod            ! lengthly data declarations/inits for main program
use shr_sys_mod         ! wrappers to system calls
use shr_timer_mod       ! timing utilities
use shr_file_mod        ! file get/put
use shr_mpi_mod         ! mpi layer

implicit none

private ! except
```

PUBLIC TYPES:

```
! none
```

PUBLIC MEMBER FUNCTIONS:

```
public :: restart_write    ! write a restart file
public :: restart_read     ! read  a restart file
public :: restart_readDate ! read  a restart file, read date only
```

PUBLIC DATA MEMBERS:

```
! none
```

23.1.1 restart_write – Create desired restart file.

Create desired restart file using input `date`. Update restart pointer file.
If `cpl_control_restNow` is false, this routine does nothing.

REMARKS:

```
Acceses data file from module data_mod .
```

REVISION HISTORY:

```
2002-Nov-06 - B. Kauffman - initial version.
```

INTERFACE:

```
subroutine restart_write(date)

  implicit none
```

INPUT/OUTPUT PARAMETERS:

```
  type(shr_date) :: date ! date associated with bundles
```

23.1.2 restart_read – read restart file

Read all data from Coupler restart file.

REVISION HISTORY:

```
2002-Nov-06 - B. Kauffman - initial version.
```

INTERFACE:

```
subroutine restart_read(date)

  implicit none
```

INPUT/OUTPUT PARAMETERS:

```
  type(shr_date),intent(inout) :: date   ! date associated with restart data
```

23.1.3 restart_readDate – read model date from restart file

Read model date from restart file and return it in output argument `cDate`.

REMARKS:

```
All processors need to read the date from the restart file.
```

REVISION HISTORY:

```
2002-Dec-13 - B. Kauffman - initial version.
```

INTERFACE:

```
subroutine restart_readDate(cDate)

  implicit none
```

INPUT/OUTPUT PARAMETERS:

```
  integer(IN),intent(out) :: cDate  ! start date from restart file
```

24 Date Logging

24.1 Module tStamp_mod – log model date and wall clock time. (Source File: tStamp_mod.F90)

Routines for logging model date, wall clock time, integration time.

REVISION HISTORY:

```
2003-Aug-28 - B. Kauffman, 1st version.
```

INTERFACE:

```
module tStamp_mod
```

USES:

```
use shr_sys_mod         ! share system routines
use cpl_kind_mod        ! kinds

implicit none

private ! except
```

PUBLIC TYPES:

```
public :: tStamp_tic

type tStamp_tic
   integer(IN) :: count ! value of hardware tic counter
   integer(IN) :: accum ! tic counts accumulated since initialization
   integer(IN) :: n     ! number of samples in accumulated count
end type tStamp_tic
```

PUBLIC MEMBER FUNCTIONS:

```
public :: tStamp_write  ! write the time stamp
```

PUBLIC DATA MEMBERS:

```
! no public data
```

24.1.1 tStamp_write - logs model date and wall clock time to stdout.

Logs model date and wall clock time to stdout using input argument `year, month, day` and `sec` and also output the average and instantaneous time difference between calls to this routine. Include `str` in output.
Generally it is expected that this routine is called periodically, eg. once per day, in which the average and instantaneous time difference info becomes quite useful.
If optional argument `tic_ext` is present, use it to determine when last call was made. If using `tic_ext`, an initial count value of less than zero implies this is the first call, thus accum and n are set zero.

REVISION HISTORY:

```
2002-aug-21 - B. Kauffman, 1st version.
```

INTERFACE:

```
subroutine tStamp_write(str,year,month,day,sec,tic_ext)
```

USES:

```
    implicit none
```

INPUT/OUTPUT PARAMETERS:

```
    character(*)    ,intent(in)             :: str     ! info text string
    integer(IN)     ,intent(in)             :: year    ! model year  (4-digits)
    integer(IN)     ,intent(in)             :: month   ! model month (2-digits)
    integer(IN)     ,intent(in)             :: day     ! model day   (2-digits)
    integer(IN)     ,intent(in)             :: sec     ! model secs  (5-digits)
    type(tStamp_tic),intent(inout),optional :: tic_ext ! external tic count data
```